UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS

TRABAJO FIN DE MÁSTER

IDENTIFICACIÓN DE PATRONES EN IMÁGENES PREPROCESADAS PARA LA DETECCIÓN DE ENFERMEDADES Y COMPORTAMIENTOS ANÓMALOS EN ANIMALES MEDIANTE BACKBONES DE REDES NEURONALES

Keisbeth Aury Matamoros Pinto

Dirigido por

David Diaz Vico

CURSO 2024 - 2025

TÍTULO: IDENTIFICACIÓN DE PATRONES EN IMÁGENES PREPROCESADAS PARA LA DETECCIÓN DE ENFERMEDADES Y COMPORTAMIENTOS ANÓMALOS EN ANIMALES MEDIANTE BACKBONES DE REDES NEURONALES

AUTOR: Keisbeth Aury Matamoros Pinto

TITULACIÓN: MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS

DIRECTOR DEL PROYECTO: David Diaz Vico

FECHA: Octubre de 2025

RESUMEN

Este Trabajo Fin de Máster aborda el desafío de la detección automatizada de patrones anómalos y enfermedades en animales mediante la aplicación de técnicas de Visión Artificial y Deep Learning. El objetivo principal es desarrollar un sistema robusto que pueda asistir en el diagnóstico precoz dentro de contextos de salud animal, etología y conservación.

Para lograrlo, se implementó un pipeline de inferencia en cascada que combina dos modelos de redes neuronales. En la primera etapa, se utiliza un modelo ConvNeXt pre-entrenado para la clasificación del tipo de animal (perro o gato). En la segunda etapa, la imagen clasificada se alimenta a un modelo de detección de objetos YOLOv11 entrenado para localizar y clasificar enfermedades cutáneas. La metodología incluyó el preprocesamiento de datasets de fuente abierta y la modificación de uno de ellos para adaptarse a la detección multiespecie.

Los resultados mostraron que el modelo ConvNeXt-Base superó a su versión Tiny en precisión para la tarea de clasificación, pero debido al gran costo computacional de este se optó por el modelo Tiny. Sin embargo, el modelo YOLOv11 presentó un rendimiento limitado en la detección (mAP50-95 $<$0.40) y una tendencia al sobrea juste, lo que se atribuye a la calidad y la heterogeneidad del dataset de entrenamiento. Se concluye que el pipeline en cascada es técnicamente factible para el diagnóstico visual automatizado, y su rendimiento final está directamente limitado por la calidad de los datos de entrenamiento y las anotaciones precisas, siendo el modelo de detección el principal punto de mejora.

Palabras clave: Visión Artificial, Deep Learning, Redes Neuronales Convolucionales, YOLO, Detección de Objetos, Inferencia en cascada (Cascading).

ABSTRACT

TITLE: *Identification of Patterns in Preprocessed Images for the Detection of Diseases and Anomalous Behaviors in Animals Using Neural Network Backbones.*

This Master's Thesis addresses the challenge of automated detection of anomalous patterns and diseases in animals through the application of Computer Vision and Deep Learning techniques. The main objective is to develop a robust system that can assist in early diagnosis within the contexts of animal health, ethology, and conservation.

To achieve this, a cascaded inference pipeline combining two neural network models was implemented. In the first stage, a pre-trained ConvNeXt model is used to classify the type of animal (dog or cat). In the second stage, the classified image is fed to a YOLOv11 object detection model trained to locate and classify skin diseases. The methodology included preprocessing open-source datasets and modifying one of them to adapt to multispecies detection.

The results showed that the ConvNeXt-Base model surpassed its Tiny version in classification accuracy, but due to the high computational cost of the former, the Tiny model was ultimately chosen. However, the YOLOv11 model exhibited limited detection performance (mAP50-95 $<$0.40) and a tendency toward overfitting, which is attributed to the quality and heterogeneity of the training dataset. It is concluded that the cascaded pipeline is technically feasible for automated visual diagnosis, and its final performance is directly limited by the quality of the training data and precise annotations, with the detection model being the main area for improvement.

Keywords: Computer Vision, Deep Learning, Convolutional Neural Networks, YOLO, Object Detection, Cascade Inference (Cascading).

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1. INTRODUCCIÓN

En las últimas dos décadas, el Deep Learning o aprendizaje profundo ha estado revolucionando una gran variedad de áreas relacionadas a la inteligencia artificial, consolidándose hoy en día como una de las tecnologías más prometedoras para procesamiento de datos complejos. Particularmente en el campo de Computer Vision o visión artificial, los avances obtenidos en redes neuronales profundas han permitido superar las limitaciones de los enfoques tradicionales, haciendo posible el realizar tareas de gran complejidad como lo son: la detección, segmentación y clasificación de objetos en imágenes y videos en tiempo real.

1.1 Contexto y justificación

Uno de los elementos clave para alcanzar estos logros ha sido el uso de Convolutional Neural Networks o redes neuronales convolucionales (CNNs), las cuales han demostrado un desempeño sobresaliente en el reconocimiento de patrones visuales. Dentro de este enfoque, el uso de redes neuronales pre-entrenadas que actúan como extractores de características visuales, mejor conocidas como Backbones, han permitido mejorar la eficiencia y la precisión de los modelos, a través de la reutilización de arquitecturas optimizadas como:

- **ResNet o Red Neuronal Residual:** se encuentran entre las mas populares y exitosas arquitecturas de backbones actualmente en uso, ya que usan conexiones residuales que le permiten a la red neuronal aprender de manera más eficiente y entrenar redes CNNs más profundas, dando como resultado una precisión mayor al mitigar el problema de desvanecimiento por gradiente (*the vanishing gradient problem*), destacándose especialmente en tareas de visión artificial [1].
- **EfficientNet:** familia de CNNs que optimizan tanto la velocidad como el tamaño del modelo, a través del método de escalamiento compuesto, el cual valga la redundancia, escala de manera uniforme tanto la profundidad, el ancho como la resolución de la red, proporcionando modelos más precisos y eficientes [1].
- **MobileNet:** familia de CNNs que ofrece modelos compactos, rápidos y eficientes en dispositivos con recursos limitados, como smartphones y sistemas embebidos, al combinar convoluciones ligeras y separables en profundidad con redes que modelan explícitamente las relaciones entre canales de características, conocidos como módulos de compresión y excitación (squeeze-and-excitation). Reduciendo de esta manera la cantidad de parámetros [1].
- **Swin Transformer:** transformador de visión jerárquico que procesa imágenes eficientemente. Introduce mecanismos de autoatención basados en ventanas que mejoran el rendimiento y la escalabilidad para imágenes de alta resolución, como fotos HDR. Surgio como mejora del ViT convencional, superando los masivos requerimientos de datos de entrenamiento e incorporando eficiencia [1].
- **ConvNext:** se encuentra entre las redes convolucionales recientes al ser una modernización de la arquitectura ConvNet (CNN) tradicional con diseños inspirados en los modelos Transformers, adquiriendo de esta manera un mejor rendimiento escalabilidad, sin perder en el proceso la simplicidad y la eficiencia que caracteriza a las CNNs. Hace uso de convoluciones en profundidad, bloques de cuellos de botella invertidos (Inverted bottleneck

blocks), así como de núcleos grandes [1].

- **YOLO:** siglas en ingles referentes a la frase "You Only Look Once", el cual es el nombre de un algoritmo de detección de objetos en tiempo real que aborda la detección como un solo problema de regresión de red neuronal convolucional (CNN), realizando todo el procesamiento en un solo recorrido a través de la red, logrando velocidades de inferencia considerablemente altas a comparación de otros modelos, lo que es esencial para aplicaciones en tiempo real [2].

1.2 Planteamiento del problema

La aplicación de estas tecnologías en el análisis de imágenes de animales tiene un potencial significativo en el ámbito de la salud, la conservación y en la investigación etológica (estudio del comportamiento animal en su entorno natural), ya que mediante la detección automática de patrones de comportamiento y anomalías visuales, es posible la identificación de enfermedades, comportamientos atípicos o incluso realizar tareas de clasificación de especies y razas, lo cual resultaría de utilidad en entornos clínicos veterinarios como en estudios ecológicos y de biodiversidad. La realización del presente Trabajo de Fin de Master se desarrollará en el lenguaje de programación Python, haciendo uso de bibliotecas especializadas como PyTorch para el diseño y entrenamiento de redes neuronales, así como Numpy, Matplotlib y Albumentations para la manipulación de datos, visualización y aumento de imágenes respectivamente.

La realización del presente *Trabajo de Fin de Master*, se desarrollará en el lenguaje de programación Python, haciendo uso de bibliotecas especializadas como *PyTorch* para el diseño y entrenamiento de redes neuronales, así como *Numpy, Matplotlib* y *Albumentations* para la manipulación de datos, visualización y aumento de imágenes respectivamente.

1.3 Objetivos del proyecto

El presente trabajo se estructurará en diversas etapas. En primer lugar, se procederá a la selección de *datasets* adecuados para el estudio que se encuentren alojados en la plataforma *Hugging Face* [3] Y *Kaggle* [4], junto con modelos *Backbone* pre-entrenados que se adapten mejor a la tarea de clasificación y detección de objetos en imágenes. Posteriormente, se realizará el procesamiento de los datos y la preparación del *dataset*, ajustando los parámetros necesarios para su correcta implementación. finalmente, se llevará a cabo el entrenamiento y ajuste del modelo, orientado a detectar objetos y a extraer patrones anómalos con fines diagnósticos o de clasificación. Esta metodología permitirá evaluar la eficacia de los modelos en tareas especificas de visión artificial, contribuyendo al avance de herramientas automatizadas para el análisis de imágenes de animales, así como para otro tipo de archivos multimedia.

1.4 Resultados obtenidos

Se realizaron pruebas de clasificación de imágenes con los modelos ConvNeXt-tiny y ConvNeXt-base sobre el dataset *animals ij5d2* [5], obteniendo mejores resultados de identificación para ConvNeXt-base a costa de un mayor costo computacional que, para entrenamientos más exhaustivos ocasionaba que le kernel cayera. En este aspecto, para equipos de bajos recursos, es más recomendable el uso de convNeXt-tiny.

Luego se realizaron pruebas de detección de objetos a través del modelo YOLOv11 [2] sobre el

dataset de enfermedades en la piel de perros [6], el cual fue modificado para agregar variedad al incorporar gatos y reduciendo la cantidad de imágenes a 400 para evitar las limitaciones del dataset debido a la mala calidad de la mayoria de las imágenes originales. Estas modificaciones, junto a la implementación de técnicas de aumentación de datos (rotación, traslación, saturación, etc) le permitieron al modelo mejorar su rendimiento con un mAP50 poco mayor a 0.50 y con un mAP50-95 menor a 0.40, lo que denota que la limitaciones del dataset hacian al modelo a tender al sobre ajuste. Sin embargo, con cada prueba el modelo mostro una clara tendencia a mejorar, por lo que creando u obteniendo un dataset de enfermedades de buena calidad no preprocesado y con anotaciones adecuadas de bounding boxes, se espera que los resultados de las métricas de rendimiento mejoren considerablemente, ya que YOLOv11 potencia particularmente los resultados de las métricas de rendimiento, lo cual será una buena adición para estudios futuros.

Finalmente, uniendo los resultados de ambos modelos para las tareas de clasificación y de detección de objetos a través del método de inferencia en cascada [7], se obtienen como resultado imagenes capaces de clasificar el tipo de animal que aparece (perro o gato), así como el tipo de enfermedad presente. En el caso de los fallos esperados, un $40\,\%$ provienen del modelo de detección, un $7{,}50\,\%$ corresponde al caso donde ambos modelos fallan, mientras un $60\,\%$ corresponde a los casos donde el modelo de detección tiene bajo porcentaje de acierto a comparación del de clasificación que suele tener 100 % en la mayoría de los casos, o incluso 70 % en algunas situaciones.

1.5 Estructura de la memoria

El ***capítulo I*** esta conformado por la introducción de proyecto, la cual comprende el contexto y la justificación de este, el planteamiento del problema, los objetivos del proyecto, un resumen de los resultados obtenidos y finalmente la estructura de la memoria que contiene una descripción resumida de cada capítulo

El ***capítulo II*** corresponde a la descripción del objetivo general, así como de los objetivos específicos, junto con una sección dedicada a los beneficios a futuro que se espera sean generados por el presente proyecto de investigación.

El ***capítulo IIi*** corresponde a los antecedentes y al estado del arte donde se incorporara todo el marco teórico del proyecto, así como los aspectos, técnicas, modelos y conceptos teóricos necesarios para el entendimiento y posteriormente para el desarrollo del plan de acción a implementar.

El ***capítulo IV*** consiste en el apartado de desarrollo del proyecto, lo que corresponde a todo el marco metodológico, así como una descripción de cada una de las fases implementadas durante el desarrollo del este.

El ***capítulo V*** comprende el apartado de resultados y de discusión, donde se reflejaran todas las pruebas realizadas con sus respectivos resultados y métricas correspondientes, así como de una discusión final orientada a los problemas presentados durante la fase de pruebas, así como de las expectativas para el proyecto y futuras investigaciones.

Finalmente los ***capítulos VI y VII*** corresponden a las conclusiones del proyecto, así como a

breves recomendaciones para futuros proyectos o líneas de investigación relacionados con esta área de estudio.

Capítulo 2. OBJETIVOS

El presente trabajo se estructurará en diversas etapas. En primer lugar, se procederá a la selección de *datasets* adecuados para el estudio que se encuentren alojados en la plataforma *Hugging Face*, junto con modelos *Backbone* pre-entrenados que se adapten mejor a la tarea de clasificación y detección de objetos en imágenes. Posteriormente, se realizará el procesamiento de los datos y la preparación del *dataset*, ajustando los parámetros necesarios para su correcta implementación. finalmente, se llevará a cabo el entrenamiento y ajuste del modelo, orientado a detectar objetos y a extraer patrones anómalos con fines diagnósticos o de clasificación. Esta metodología permitirá evaluar la eficacia de los modelos en tareas especificas de visión artificial, contribuyendo al avance de herramientas automatizadas para el análisis de imágenes de animales, así como para otro tipo de archivos multimedia.

2.1 Objetivos generales

Detectar patrones y comportamientos anómalos en *datasets* de imágenes de animales obtenidos desde la plataforma *Hugging Face*, mediante la implementación de *Backbones* y métodos de *Deep Learning* orientados a la visión artificial para tareas de detección de objetos.

2.2 Objetivos específicos

1. Seleccionar *datasets* de imágenes y modelos *Backbone* pre-entrenados adecuados para implementar en tareas de visión artificial aplicadas al análisis de imágenes de animales.
2. Establecer los parámetros óptimos y realizar el preprocesamiento de los datos para preparar adecuadamente el conjutno de entrenamiento y validación, adaptándolo al *Backbone* seleccionado.
3. Entrenar y refinar el modelo de detección de objetos, con el objetivo de identificar patrones visuales y anomalías en las imágenes de animales, orientado a la detección de enfermedades, comportamientos atípicos y clasificación por especie y raza.

2.3 Beneficios del proyecto

Los avances metodológicos desarrollados podrán ser aprovechados en investigaciones futuras por otros investigadores o grupos de investigación, o incluso en asignaturas que estén relacionadas con el ámbito de la ingeniería, de la inteligencia artificial y ciencias aplicadas. Además dado que el presente proyecto se centra en la detección de enfermedades y comportamientos anómalis en animales mediante tecnologia accesible, el proyecto tambien puede llegar a inspirar iniciativas con valor ético, social y ecológico, sensibilizando a estudiantes hacia investigaciones futuras con impactos reales.

Capítulo 3. ANTECEDENTES / ESTADO DEL ARTE

3.1 Estado del arte

El cerebro humano es un órgano sumamente complejo, el cual se encuentra principalmente compuesto por neuronas (células fundamentales del sistema nervioso), las cuales son las encargadas de transmitir información desde el cerebro a cualquier otra parte del cuerpo. De hecho, las neuronas junto a las llamadas *células gliales*, forman redes complejas que permiten al cerebro realizar funciones como el pensar, sentir, controlar el movimiento, aprender, entre muchas otras funciones tanto sencillas como de mayor grado de complejidad.

En la literatura, es común el comparar a estas redes neuronales con circuitos eléctricos debido a la manera en la que transmiten, reciben y procesan información a través de impulsos eléctricos. Esta comparación ha sido de utilidad tanto dentro del ámbito científico como médico para entender mejor el funcionamiento y comportamiento de las neuronas y de los llamados circuitos neuronales. Teniendo esto en cuenta, no es de extrañar que surgiese la idea de desarrollar sistemas artificiales inspirados en el funcionamiento del cerebro humano y sus neuronas, para realizar tareas complejas en el ámbito computacional.

3.2 Contexto y justificación

En este contexto, una **red neuronal artificial** (*Artificial Neural Network or ANN*) es un sistema de procesamiento de información o de señales, el cual se encuentra compuesto por una gran cantidad de elementos de procesamiento simples y lineales interconectados por enlaces directos, los cuales cooperan para realizar procesamientos distribuidos complejos, con la finalidad de resolver determinadas tareas computacionales. Ahora, en lo que refiere a la computación convencional, puede decirse que las redes neuronales adquieren un enfoque diferente en lo que respecta a la resolución de problemas, ya que los sistemas computacionales convencionales hacen uso de un enfoque de resolución basado en algoritmos, por ejemplo, como lo es el seguir un conjunto de pasos o instrucciones para resolver un problema, lo cual limita el proceso de resolución de problemas a aquellos que entiende y que ya sabe resolver [8].

Sin embargo, en lugar de hacer una comparación competitiva entre los algoritmos computacionales convencionales y las redes neuronales, se debe tener en cuenta que ambos métodos se complementan, ya que existen tareas para las que se adapta mejor un enfoque algorítmico (por ejemplo, operaciones aritméticas y similares), así como existen otros tipos de tareas para las cuales se adapta mejor un enfoque basado en redes neuronales, como lo son las tareas de visión artificial, modelos de lenguaje natural, entre otras.

Entre las tareas para las que suele ser más conveniente un enfoque basado en redes neuronales, se encuentran:

- **Visión artificial (Computer Vision).** Comprende las tareas de clasificación (etiquetar imágenes), detección de objetos (ubicar y clasificar objetos), segmentación de imágenes (asignación de etiquetas a cada píxel), recuperación de imágenes (encontrar imágenes

relevantes en bases de datos asociadas a consultas), entre otras [9].

- **Procesamiento de lenguaje natural (Natural Language Processesing NLP).** Comprende modelado de lenguaje, procesamiento morfológico, análisis sintáctico y procesamiento semántico [10]. Entre los más conocidos se encuentran: GPT (desarrollado por OpenAI) para procesamiento de texto, BERT (desarrollado por Google) para compresión de contexto bidireccional, entre otros.
- **Audio y procesamiento de señales.** Comprende reconocimiento de voz (Speech Recognition or Automatic Speech Recogntion ASR), sintetizador de voz (Text-To-Speech) e identificación del habla (Speaker Recognition).
- **Series temporales y predicción.** Comprende, la regresión, predicción de series temporales, detección de anomalías, entre otras. Donde cabe destacar el uso de redes neuronales para la predicción de series temporales, usando tamaños de muestreo y de ventana (comúnmente usadas en las predicciones de mercado, previsiones meteorológicas, pronóstico del tráfico de red, etc.) adecuados, como lo señala [11].
- **Aprendizaje por refuerzo / juegos / control.** Comprende juegos, planificación, toma de decisiones, robótica y control de movimiento entre otras. Un ejemplo es el uso de redes neuronales basadas en el aprendizaje por refuerzo para el evitamiento de obstáculos en robots móviles [12].
- **Sistemas de recomendación.** Consiste en la recomendación personalizada de productos, series, etc. Los sistemas de recomendación más utilizados suelen generar una lista de recomendaciones a través del filtrado colaborativo o basado en el contenido. Ninguno de estos enfoques tiene en cuenta el contenido de las reseñas escritas, que contienen información valiosa sobre los gustos de los usuarios, por lo que hay enfoques basados en RNNs [13], así como enfoques híbridos que implementan ANNs y métodos como el descenso por gradiente [14], entre otros.
- **Tareas multimodales.** Comprenden la combinación de tareas de visión, texto y audio como lo son el generar texto a partir de imágenes, responder preguntas sobre imágenes, o conversaciones multimodales (voz más texto, más imágenes) entre otras [15].

Cabe destacar que actualmente, lo métodos para resolver problemas relacionados con este tipo de tareas, siguen evolucionando y mejorando continuamente, impulsando modelos de IA populares hoy en día como lo son Gemini, ChatGPT, entre otros. Sin embargo, el enfoque de interés del presente trabajo se encuentra relacionado con el uso de redes neuronales, así como en la aplicación de técnicas de aprendizaje profundo (Deep Learning) para la resolución de problemas asociados a tareas de visión artificial, particularmente en tareas de detección de objetos para la determinación de patrones y comportamientos anómalos en imágenes de animales.

Sin embargo, antes es necesario comprender mejor este tipo de procesos, iniciando con la Visión artificial (Computer Vision), la cual se encuentra relacionada con el procesamiento de imágenes. Sin embargo, no se deben confundir entre sí, ya que no se trata del mismo método, pero con nombres diferentes, dado que el procesamiento de imágenes es un proceso de ***bajo nivel*** que se centra en la manipulación y mejora de imágenes, enfocándose específicamente en realizar acciones como variación del contraste, filtrar ruido, detectar bordes, redimensionar,

aplicar negativos, saturación etc. Mientras que el computer visión es un proceso de ***alto nivel*** que se enfoca en entender el contenido de la imagen usando el procesamiento de imágenes como un paso previo, pero yendo más allá ya que su enfoque consiste en que las computadoras sean capaces de "ver" e interpretar archivos multimedia como imágenes y videos.

Teniendo esto en cuenta, de manera similar a la manera en la que las redes neuronales se inspiran en las redes neuronales del cerebro humano, se puede decir que el computer vision funciona de manera similar a la visión del ser humano, el cual a través de sus ojos que perciben la información visual, que luego pasa a ser procesada por la retina en forma de impulsos nerviosos que viajan hasta la corteza visual, ubicada en la parte posterior del cerebro y la cual se encarga de la percepción y de la interpretación de las imágenes recibidas, lo que en conjunto con la gran cantidad de información contextual que el ser humano acumula a lo largo de su vida, el sistema visual humano mejora siendo capaz de distinguir entre objetos, estimar su distancia, detectar movimiento así como patrones e incluso anomalías en imágenes. El computer vision es capaz de llevar a cabo estas tareas a través del uso de cámaras u otros dispositivos, de la mano con análisis estadísticos y el uso de algoritmos adecuados, denotando así una gran diferencia respecto al ser humano al ser capaz en algunos casos de realizar todo lo antes mencionado con mayor eficiencia y rapidez, pudiendo incluso de manera más minuciosa identificar problemas o detalles que podrían no ser fácilmente apreciables por el ojo humano [16].

En este contexto los objetivos principales de esta área de estudio implican la recopilación de datos sobre acciones o descripciones, como lo son la captura de escenas y la extracción o detección de objetos. No obstante, es importante tener en consideración que la selección del enfoque a utilizar para la resolución de problemas en esta área depende del dominio operativo especifico, así como de las propiedades inherentes de los datos a analizar. Para finalizar, cabe destacar que, entre los enfoques de resolución más comunes se encuentran los siguientes:

- ***Clasificación:*** este enfoque consiste en identificar los elementos presentes ya sea en imágenes o videos, asignando (una o más) etiquetas o clases generales identificativas acordes a los elementos presentes (personas, gato, football, noticias, película, etc.) en el archivo multimedia. Entre los ejemplos de uso están el de detectar violencia en un video, analizar imágenes médicas, o incluso detectar lo que hay alrededor de un vehículo, etc. Ahora, es importante tener en cuenta que, dependiendo de la probabilidad de la presencia visual de un determinado objeto, puede ocurrir que se etiquete solamente la clase de dicho objeto, lo que puede implicar perdida de información. Sin embargo, a través de la codificación dispersa (Sparse Coding), es posible recuperar la perdida de información ocurrida durante el proceso clasificación [17].
- ***Detección de objetos:*** este enfoque va un paso más allá de la clasificación, ya que se caracteriza por ser más específico al identificar el tipo del objeto y donde se encuentran en el archivo multimedia, encerrando a cada objeto en cajas delimitadoras (Bounding Boxes) con su respectiva etiqueta. El proceso consiste en tomar una imagen o video como entrada (input) y dependiendo de la clase y de la posición se puede realizar una determinación del/los objeto(s) presentes, siendo PASCAL VOC (nombre de un conjunto de datos o dataset) con veinte clases uno de los conjuntos más usados en tareas de

detección o identificación de objetos [9].

- ***Recuperación de imágenes:*** el cual, a grandes rasgos se puede definir como la tarea de encontrar imágenes relevantes en una base de datos en respuesta a una consulta, la cual puede ser realizada a través de texto o de inclusive otra imagen. Este tipo de procesos son un ejemplo del cómo el empleo de redes neuronales (particularmente de los modelos convolucionales), pueden ofrecer un rendimiento mejor que el de métodos más convencionales. Se deriva del concepto de pirámide espacial (Spatial Pyramid), dónde sin eliminar ningún tipo de información espacial de la imagen, toma la imagen principal y la divide en diferentes escalas, incluso es posible el entrenar un gran conjunto de datos tanto para la extracción de características como para la recuperación de imágenes [9].
- ***Segmentación semántica:*** La cual, a grandes rasgos puede decirse que consiste en asignar una etiqueta a cada píxel de una imagen. Para este tipo de tarea se usan redes neuronales convolucionales ya que son capaces de trabajar las predicciones realizadas por los modelos generados al nivel del píxel, además la segmentación semántica requiere la implementación de máscaras de salida (Output masks) para realizar alguna de las siguientes tareas:
  - ***Segmentación basada en detección:*** Este enfoque divide la imagen dependiendo de los resultados de la detección de objetos [9].
  - ***Segmentación basada en FCN-CRFs:*** En este tipo de enfoque la capa completamente conectada (Fully Conected Network FCN) es reemplazada por las capas completamente convolucionales, lo cual es un método comúnmente usado para segmentación semántica. Se modelo similar es propuesto por las redes completamente convolucionales (Fully Convolutional Networks FCN) integradas con campos aleatorios convolucionales (Convolutional Random Fields CRFs) para la recuperación de limitas o cotas elaboradas [9].
  - ***Anotaciones débilmente supervisadas:*** Este enfoque, además de mejorar el modelo de segmentación, toma la segmentación de los datos en entrenamiento, en función de los cuales estima las máscaras de segmentación para actualizar la red neuronal, luego combina los pixeles pequeños para obtener la imagen completa [9].

A continuación, considerando las características y enfoques antes mencionados, surge la siguiente pregunta ¿Qué se necesita para poder resolver problemas relacionados con esta tarea? Para empezar, es necesario conocer un poco sobre aprendizaje automático (Machine Learning) también, la cual es un área de la inteligencia artificial (IA) caracterizada por enfocarse en la creación de sistemas que poseen la capacidad de aprender de manera automática a partir de datos suministrados, sin la necesidad de ser programados explícitamente para realizar tareas específicas. La manera de lograr este tipo de desempeño difiere de la computación convencional basada en algoritmos y en el desarrollo de un conjunto de reglas específicas, ya que el machine learning se centra en la creación y en el entrenamiento de modelos para que estos sean capaces de reconocer patrones en los datos, para luego realizar predicciones basadas en los datos de muestra. Es decir, implica el "enseñarle" al ordenador a aprender de los datos para luego poder realizar predicciones o tomar decisiones por su cuenta.

Por otra parte, se encuentra también el ***aprendizaje profundo (Deep Learning)***, el cual es una subárea del machine learning, que se caracteriza principalmente por enfocarse en el uso de ***redes neuronales profunda***s, es decir, de redes neuronales con una gran cantidad de capas (***layers***), de donde proviene la idea de profundidad en el método, dado que, a mayor cantidad de capas, entonces más "profunda" es la red. El deep learning, de manera similar a su predecesor, entrena modelos computacionales para que aprendan automáticamente a reconocer patrones en grandes volúmenes de datos (***Big Data***). Este proceso lo realiza a través de varias capas de procesamiento, mediante las cuales los modelos son capaces de extraer representaciones de los datos cada vez más complejas.

Es esta capacidad de representar datos de manera distribuida con múltiples capas de abstracción, lo que ha impulsado la popularidad del deep learning, por lo que se ha ampliado la gama de algoritmos de deep learning que se emplean para resolver incluso problemas convencionales de inteligencia artificial. Particularmente, el deep learning tiene mucho campo de uso en las áreas de procesamiento de imágenes y videos, así como en visión artificial y en la bioinformática, entre otros.

Algunos de los modelos de deep learning son, entre otros, los siguientes [9]:

- ***Redes neuronales convolucionales (Convolutional Neural Network, CNN).***
- ***Máquinas de Boltzmann Restringidas (Restricted Boltzmann Machine).***
- ***Autoencoders.***

Cada una tiene sus características distintivas, así como sus ventajas y desventajas dependiendo de la tarea o problema que se pretenda resolver. En el caso de computer vision, las CNNs se encuentra entre los modelos con mejor desempeño en general, tanto para datasets pre-entrenados como para los que no [18].

Teniendo esto en cuenta, es conveniente entender que es CNN y cómo se comporta.

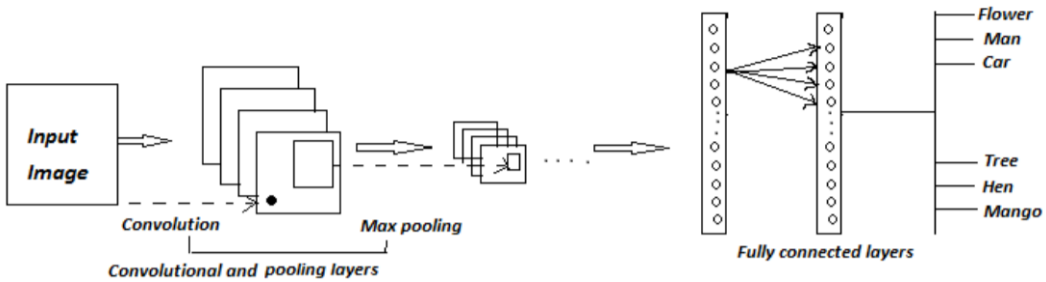

Figura 3.1. Arquitectura en Pipeline de una red neuronal convolucional básica. Fuente: árticulo *Deep Learning For Computer Vision Tasks: A review* [9]

La arquitectura de la CNN mostrada en la figura 1, consiste principalmente en tres tipos de capa [9]:

- **Capas convolucionales:** En esta capa, la CNN usa una gran cantidad de filtros para convolucionar toda la imagen, los mapas de características intermedios y para generar diferentes mapas de características. Durante este proceso se logra la reducción de pará-

metros mediante la distribución de pesos, también la correlación entre píxeles vecinos es más fácil debido a la conectividad local; así como se mantiene la ubicación del objeto fija.

- **Capas de agrupamiento (pooling):**similar a la anterior solo que reduce las dimensiones de los mapas de características y también los parámetros de la red.
- **Capas completamente conectadas:** contiene el 90 % de los parámetros. La red feed forward forma un vector de una longitud específica para continuar con el procesamiento. Dado que estas capas contienen la mayoría de los parámetros, existe una alta carga computacional durante el entrenamiento de los datos.

El entrenamiento de la CNN puede dividirse en etapas hacia adelante y hacia atrás. En la etapa hacia adelante, primero se clasifica la imagen de entrada según sus pesos y sesgos para cada capa. El coste de pérdida (***loss***) se calcula a partir de los datos de entrada usando la salida predicha. En la etapa hacia atrás (***Backward***), dependiendo del coste de pérdida medido, se calculan los gradientes para cada parámetro. Luego, a través de estos gradientes, se actualizan los parámetros para la siguiente iteración. Finalmente, el procedimiento de entrenamiento puede detenerse tras un número considerable de repeticiones, en las que ya no se aprecie mejora para evitar el sobre ajuste del modelo [9].

Por otra parte, como toda red neuronal, las CNNs tienen una estructura central y en deep learning orientado a la resolución de tareas de computer vision al igual que en modelización de lenguaje, esta parte central se denomina ***Backbone***, el cual funciona como el extractor de características principal de los datos de entrada (***input data***) de un modelo. Comúnmente las redes neuronales suelen estar construidas sobre un backbone, el cuál puede estar pre-entrenado o no. En la decada de los $2010s$ se solía usar una base de datos de imágenes organizada acorde a la jerarquía WordNet, en la que cada nodo de la jerarquía se encontraba representado por cientos y miles de imágenes, dicha base de datos es conocida como ImageNet y además de servir de apoyo para el avance de la investigaciones en computer vision y en deep learning, también solía usarse para pre-entrenar una red convolucional, pero recientemente han comenzado a aparecer cada vez más backbones pre-entrenados con diferentes datesets y algoritmos, así como con considerables mejoras en su rendimiento, lo cual ha complicado gradualmente la tarea de escoger un backbone adecuando al problema que se pretende resolver entre la gran variedad que hay actualmente, es por ello que se ha usado de guía el articulo Battle of Backbones [18], el cual realiza una extensa comparación entre diversos backbones pre-entrenados probados en distintas tareas como clasificación, detección de objetos, segmentación, generalización fuera de distribución (***Out-Of-Distribution OOD generalitation***) y recuperación de imágenes, obteniendo los siguientes resultados:

Task	Good	Better	Best
1 Cls	ConvNeXt-B (IN-21k)	CLIP ViT-B (LAION-2B)	Sup. SwinV2-B (IN-21k,1k)
2 Det	Sup. ConvNeXt-B (IN-1k)	Sup. SwinV2-B (IN-21k,1k)	Sup. ConvNeXt-B (IN-21k)
3 Seg	Sup. ConvNeXt-B (IN-1k)	Sup. SwinV2-B (IN-21k,1k)	Sup. ConvNeXt-B (IN-21k)
4 Ret	CLIP ViT-B (LAION-2B)	Sup. SwinV2-B (IN-21k,1k)	Sup. ConvNeXt-B (IN-21k)
5 (OOD) Cls	CLIP ViT-B (LAION-2B)	Sup. SwinV2-B (IN-21k,1k)	Sup. ConvNeXt-B (IN-21k)
6 (OOD) Det	Sup. ConvNeXt-B (IN-21k)	Sup. ConvNeXt-T (IN-1k)	Sup. ConvNeXt-B (IN-1k)
7 All	CLIP ViT-B (LAION-2B)	Sup. SwinV2-B (IN-21k,1k)	Sup. ConvNeXt-B (IN-21k)

Figura 3.2. Se muestran los tres backbones con mejor rendimiento (de izquierda a derecha) para varias tareas y configuraciones. El rojo corresponde a las evaluaciones fuera de distribución (OOD) y el verde indica comparaciones generales. Fuente: articulo *Battle of Backbones* [18]

Como se puede apreciar en la tabla anterior (Fig 2), en general la red neuronal convolucional supervisada ConvNeXt-B, entrenada sobre el conjunto IN-21k (ImageNet-21k, con $21k = 21{,}000$ clases o categorías), es la que presenta el mejor rendimiento para cada tipo de tarea, como muestra la fila 7 de la tabla en la columna "Best".

Ahora, esto es una comparación general, empleando backbones potentes y datasets con una enorme cantidad de datos, que pueden ser soportados por equipos (ordenadores) con una considerable capacidad de cómputo, pero ¿Qué ocurre en el caso contrario? ¿Qué sucede si no se cuentan con los recursos suficientes para implementar alguno de estos backbones? Pues en ese caso, el artículo Battle of Backbones [18] proporciona la siguiente comparación:

Task	Good	Better	Best
1 Cls	ResNet-18	RegNetX-400MF	EfficientNet-B0
2 Det	RegNetX-400MF	EfficientNet-B0	ResNet-18
3 Seg	RegNetX-400MF	EfficientNet-B0	ResNet-18
4 Ret	ResNet-18	RegNetX-400MF	EfficientNet-B0
5 (OOD) Cls	ResNet-18	RegNetX-400MF	EfficientNet-B0
6 (OOD) Det	EfficientNet-B0	ResNet-18	RegNetX-400MF
7 All	ResNet-18	RegNetX-400MF	EfficientNet-B0

Figura 3.3. Clasificación de backbones ultra livianos con mejor rendimiento (de izquierda a derecha) para varias tareas y configuraciones. El rojo corresponde a las evaluaciones fuera de distribución (OOD) y el verde indica las comparaciones generales. Fuente: articulo *Battle of Backbones* [18]

Como se puede apreciar en la tabla anterior, ***ResNet-18*** es una red con buen rendimiento en todos los casos, mientras que ***EfficienteNet-B0*** es la que tiene mejor desempeño en general.

Como el problema a resolver en el presente trabajo de fin de máster, se encuentra relacionado con la detección de objetos, se tomará en consideración los resultados de las dos tablas anteriores, las cuales destacan a dos candidatos por encima de los demás, ResNet-18 para bajos recursos y ConvNeXt-B entrenado en IN-21k para datasets enormes y equipos con altos recursos. Donde incluso se puede agregar otra opción ConvNeXt para recursos moderados o bajos, el cual es ***ConvNeXt (tiny-sized model)*** desarrollado por ***Facebook AI Research***, cargado en la web ***Huggin Face*** como ***facebook/convnext-tiny-224*** y presentado en el artículo "A

ConvNet for the 2020s[19]

Este modelo forma parte de la familia ***ConvNeXt***, la cual es una línea de redes convolucionales "modernizadas" inspiradas por los Vision Transformers. En particular, el diseño del modelo Tiny adapta la arquitectura de ResNet, actualizada con ideas de Swin Transformer, para ofrecer un backbone eficiente y de alto rendimiento, entrenado sobre ImageNet-1k con imágenes de resolución 224×224, aunque también se puede trabajar con datasets de imágenes con diferentes tipos de resoluciones.

Ahora, teniendo en cuenta la métrica de la prueba de perdida de entropía cruzada ***(Cross-entropy test loss CE)***, la cual puede expresarse matemáticamente a través de la siguiente ecuación:

$$CE = \frac{1}{N}\Big[\sum_{j=1}^{N}[t_j log(p_j) + (1-t_j)log(1-p_j)]\Big]$$

para N puntos de datos donde t_i es el valor de la verdad y p_i es la probabilidad Softmax para el punto i^n.

Luego, considerando el siguiente rango de valores posibles:

- ***Valores cercanos a 0***, implican que la interpretación del modelo es excelente, ya que predice muy bien las etiquetas, posee una certeza y precisión altas.
- ***Valores entre 0.1 a 0.5***, implican que la interpretación del modelo es buena, ya que tiene buen desempeño, aunque acepta algo de error o incertidumbre.
- ***Valores entre 0.5 a 1.0***, implican que la interpretación del modelo es regular, ya que el modelo tiene margen para mejorar, porque aún hay errores significativos.
- ***Valores mayores a 1.0***, implican que el modelo cuenta con una mala predicción y la distribución no coincide con etiquetas.

Y que para la métrica del ***Error de calibración esperado (Expected Calibration Error ECE***)

$$ECE = \Big[\sum_{j=1}^{N}\frac{|B_m|}{n}|accuracy(B_m) - confidence(B_m)|\Big]$$

La medida implica dividir los datos en M contenedores (o "bins") espaciados equitativamente. Se utiliza B para representar ¸contenedoresz m para el número de contenedor. Luego, los valores de la ECE son los siguientes:

- ***Valores entre 0 % a 1 %***, implican que la interpretación del modelo es excelente y que este se encuentra muy bien calibrado, ya que la confianza coincide con la precisión real.
- ***Valores entre 1 % a 3 %***, implican que la interpretación del modelo es buena, ya que la calibración es buena y el modelo es confiable para tomar decisiones.
- ***Valores entre 3 % a 7 %***, implican que la interpretación del modelo es regular, ya que la calibración es moderada y el modelo puede sobre- o subestimar la confianza.

- ***Valores mayores a 7 %***, implican que el modelo cuenta con una mala interpretación, ya que está mal calibrado y puede dar confianza errónea (ya sea muy optimista o pesimista).

Entonces, considerando los resultados obtenidos en la siguiente tabla del árticulo Battle of Backbones, se podría decir que el modelo ConvNeXt (tiny-sized model) pareciera ser la combinación o el punto intermedio entre los candidatos que resalta el árticulo (ResNet y ConvNeXt-B), acorde a sus valores de precisón (Accuracy), CE y ECE.

Backbone	Method	Pretrain Data	Accuracy	CE	ECE
ResNet-50	Supervised	ImageNet-1k	80.38	0.94	0.09
	VicReg	ImageNet-1k	78.77	1.11	0.21
	CLIP	LAION-2B	80.55	1.02	0.19
	DINO	ImageNet-1k	79.08	1.11	0.22
ResNet-101	Supervised	ImageNet-1k	81.93	0.92	0.16
ViT-Small	Supervised	ImageNet-1k	78.84	0.84	0.03
	Supervised	ImageNet-21k	81.39	0.68	0.01
	MoCoV3	ImageNet-1k	79.65	0.90	0.11
	DINO	ImageNet-1k	81.33	0.83	0.10
ViT-Base	Supervised	ImageNet-1k	79.15	0.86	0.05
	Supervised	ImageNet-21k	84.53	0.56	0.01
	MoCoV3	ImageNet-1k	82.85	0.77	0.08
	MAE	ImageNet-1k	83.41	0.75	0.09
	DINO	ImageNet-1k	83.40	0.76	0.07
	CLIP	LAION-2B	85.19	0.66	0.08
SwinV2-Tiny	Supervised	ImageNet-1k	81.82	0.83	0.09
	MiDaS	MiDaS	82.55	0.83	0.07
ConvNeXt-Tiny	Supervised	ImageNet-1k	82.10	0.79	0.06
ConvNeXt-Base	Supervised	ImageNet-1k	83.88	0.69	0.04
	Supervised	ImageNet-21k	85.87	0.56	0.03

Figura 3.4. Calibración y estimación de incertidumbre en ImageNet. Las métricas utilizadas son: la precisión Top-1 (%), la pérdida de prueba de entropía cruzada (Cross-Entropy CE) y el error de calibración esperado (ECE) Fuente: articulo *Battle of Backbones* [18]

Sin embargo, aunque el modelo ConvNeXt (tiny-sized model) parece ser el backbone adecuado para la tarea a realizar, ya que no demanda una excesiva cantidad de recursos y tiene un buen desempeño en general. No obstante, no conviene descartar aún los modelos ResNet y ConvNeXt, ya que alguno podría mostrar mejor desempeño que el resto en lo que respecta al problema a resolver. Dicho esto, solo falta orientar el backbone escogido a que se especialice o se encargue de una tarea especifica, que, en este caso, sería la detección de objetos, para la cual se puede emplear un dataset pre-entrenado para esta tarea, lo que aseguraria la presencia de las bounding boxes y agilizaria el proceso de entramiento; de lo contrario otra opción viable consistiría en la preparación manual de dataset a utilizar, procesando y adecuando las imágenes antes de realizar las anotaciones de las coordenadas que conforman las bounding boxes, con herramientas como makesense.ai [20], label-studio [21], entre otras.

Por otro lado, conviene considerar el algoritmo de detección de objetos **YOLO** (***You Only Look Once***), el cual se destaca por su velocidad y eficiencia respecto a otros modelos y redes convolucionales. A diferencia de otros anteriores que utilizaban un proceso de dos pasos (primero proponer regiones y luego clasificar), YOLO aborda la detección como un solo problema

de ***regresión de red neuronal convolucional (CNN)***, a través de los siguientes pasos [2].

- **División en Cuadrícula:** La imagen de entrada se divide en una cuadrícula de tamaño N×N.
- **Localizada:** Cada celda de la cuadrícula es responsable de detectar objetos cuya parte central cae dentro de ella.
- **Predicción Única:** Cada celda predice simultáneamente lo siguiente:
  - Múltiples cuadros delimitadores (bounding boxes) con sus coordenadas.
  - Un valor de confianza, que representa la probabilidad de que el cuadro contenga un objeto.
  - Probabilidades de clase (qué objeto es).

Al realizar todo el procesamiento en un solo recorrido a través de la red, YOLO logra una velocidad de inferencia excepcionalmente alta, lo que es particularmente útil y esencial para aplicaciones en tiempo real (como vehículos autónomos o robótica).

YOLOv11 es la última generación desarrollada por Ultralytics [2], la empresa detrás de YOLOv5, v8, etc. La cual representa el estado del arte en detección y tareas de visión por computadora. Esta versión se caracteriza por ser capaz de adecuarse a múltiples tareas, como lo son la clasificación, la detección de objetos orientada (***OBB or Oriented Bounding Boxes***), la segmentación de imágenes y la estimación de poses. Además, es capaz de lograr mayor precisión mAP con menor cantidad de parámetros y mayor velocidad de inferencia que versiones anteriores.

Variante	Nombre	Caracteristica clave	Uso
YOLOv11n	Nano	El más rápido y pequeño. Mínimo de parámetros.	Aplicaciones en tiempo real con recursos limitados (ej. dispositivos de borde, CPUs). Sacrifica algo de precisión.
YOLOv11s	Small	Buen equilibrio entre velocidad y precisión.	Uso general que requiere un rendimiento rápido sin grandes sacrificios de mAP.
YOLOv11m	Medium	Un modelo robusto con un buen equilibrio de rendimiento.	La opción por defecto para muchos proyectos. Ofrece alta precisión y velocidad moderada.
YOLOv11l	Large	Mayor profundidad y anchura de red.	Ideal cuando la precisión es crítica. Más lento que las variantes pequeñas.
YOLOv11x	Extra-Large	El más preciso y complejo. Máximo de parámetros.	Usado cuando se requiere la máxima precisión (mAP). Es el más lento y consume más recursos computacionales (GPU potente).

Figura 3.5. Variaciones de la versión YOLOv11, deseñados para adaptarse a diferentes necesidades y cuya principal diferencia radica en el tamaño y en el ancho de red de cada variación. Fuente: *Ultralytics* [2]

Una vez obtenido (o armado) el dataset, se puede lograr orientar el backbone para detección de objetos a través de un método conocido como ajuste fino (***finetuning***), el cual consiste en tomar un modelo o backbone previamente entrenado y entrenarlo nuevamente con datos nuevos que generalmente son más pequeños o especializados, adaptándolo así a una tarea especifica.

El proceso del finetuning consiste en los siguientes pasos:

1. Tomar un backbone previamente entrenado.

2. Reemplazar o ajustar las últimas capas para la nueva tarea (por ejemplo, cambiar la capa de salida para nuevas clases).

3. Entrenar el modelo con un conjunto de datos específico, usualmente con una tasa de aprendizaje más baja para no "olvidar" lo aprendido.

Entre los tipos de finetuning, se encuentran:

- **Fine-tuning completo (Full fine-tuning):** Se reentrena todas las capas del modelo pre-entrenado con el nuevo conjunto de datos, lo que le permite al modelo ajustar todas sus características para la nueva tarea y a pesar de que requiere más datos y recursos, puede ofrecer un mejor rendimiento si se realiza de la manera adecuada [22].
- **Fine-tuning completo (Full fine-tuning):** Se reentrena todas las capas del modelo pre-entrenado con el nuevo conjunto de datos, lo que le permite al modelo ajustar todas sus características para la nueva tarea y a pesar de que requiere más datos y recursos, puede ofrecer un mejor rendimiento si se realiza de la manera adecuada [22].
- **Fine-tuning completo (Full fine-tuning):** Se reentrena todas las capas del modelo pre-entrenado con el nuevo conjunto de datos, lo que le permite al modelo ajustar todas sus características para la nueva tarea y a pesar de que requiere más datos y recursos, puede ofrecer un mejor rendimiento si se realiza de la manera adecuada [22].
- **Fine-tuning parcial (Partial fine-tuning):** Solo se reentrena un subconjunto de capas, normalmente las capas superiores, que son las más cercanas a la salida (output). Luego las capas inferiores, que son las más cercanas a la entrada (input) se congelan (es decir, no se actualizan). Este proceso es especialmente útil cuando hay pocos datos o cuando las primeras capas extraen características generales que no necesitan ajustarse [15].
- **Fine-tuning de cabeza (Head fine-tuning):** Solo se entrena la última capa o "cabeza" del modelo, que sería la capa de clasificación o de salida. El resto del modelo permanece congelado. Este sería el método más rápido y eficiente en datos pequeños, pero con menor flexibilidad [22].
- **Fine-tuning escalonado (Layer-wise fine-tuning o gradual unfreezing):** Se descongelan y reentrenan las capas gradualmente, comenzando desde las últimas capas hacia las primeras, lo cual permite un ajuste más controlado y evita que el modelo olvide lo aprendido. Este método se usa especialmente en NLP y transfer learning [15].
- **Fine-tuning con regularización (Regularized fine-tuning):** Se aplican técnicas de regularización (dropout, weight decay, etc.) durante el ajuste fino para evitar sobre ajuste. Además, tiene como ventaja que puede combinarse con cualquiera de los métodos anteriores [22].

Finalmente, tras afinar el modelo, es necesario evaluarlo para poder medir el rendimiento real del mismo usando métricas adecuadas para cada tarea (clasificación, detección, etc).[23].

Por otra parte, en lo que respecta a ***clasificación*** se debe tener en cuenta que las métricas más implementadas son las siguientes:

- ***Accuracy (exactitud o precisión global):*** es la proporción de todas las predicciones (clasificaciones) correctas, sobre el total de predicciones realizadas, lo cual se puede

expresar matemáticamente de la siguiente manera:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

donde,

- **TP:** True Positive / verdaderos positivos.
- **TN:** True Negative / verdaderos negativos.
- **FP:** False Positive / falsos positivos.
- **FN:** False Negative / falsos negativos.

Dado que esta métrica incorpora los cuatro resultados posibles, si ***el dataset es equilibrado (clases balanceadas),*** el accuracy funciona como medida de calidad del modelo de baja granularidad, razón por lo cual suele ser la métrica de evaluación "predeterminada" para modelos que realizan tareas genéricas o no especificadas. Sin embargo, si el conjunto no está equilibrado, o uno de los errores FP o FN es más costoso que el otro, como suele suceder en la mayoría de los casos del mundo real, entonces el accuracy por sí solo no es suficiente para medir el rendimiento del modelo.

- ***Precision (por clase):*** se define como la proporción de todas las predicciones positivas que realmente lo son, por lo que sirve para determinar qué tan precisas son las predicciones positivas. La precisión, puede expresarse matemáticamente de la siguiente manera:

$$Precision = \frac{TP}{TP + FP}$$

La precisión mejora a medida que disminuyen los falsos positivos. En datasets con datos desequilibrados, donde la porción de predicciones positivas es muy baja, la precisión es menos significativa y útil como métrica.

- ***Recall (por clase):*** también conocida como recuperación, muestra la proporción de todos los positivos reales que se clasificaron correctamente como positivos, por lo que esta métrica sirve para determinar que acertado es el modelo para detectar los positivos reales, además puede expresarse matemáticamente de la siguiente manera:

$$Recall = \frac{TP}{TP + FN}$$

En este caso se toman en cuenta los FN, ya que son positivos reales que se clasificaron erróneamente como negativos. En un conjunto de datos desequilibrado con una cantidad de positivos reales muy baja, el recall como métrica es más significativa que la precisión, ya que mide la capacidad del modelo para identificar correctamente todos los positivos reales. En este aspecto se puede apreciar como Recall tienen una relación inversa, por lo que se puede deducir que métodos para mejorar uno, ocasionaran que el otro empeore

proporcionalmente.

- ***F1-score (por clase):*** es la media armónica entre la precisión y el accuracy, la cual es especialmente útil cuando el conjunto está desequilibrado o hay clases desbalanceadas. Como la media armónica tiende a mitigar el impacto de los grandes valores atípicos y a agravar el impacto de los pequeños. Es decir, el F1-score castiga fuertemente los valores extremos. Matemáticamente se puede expresar esta métrica a través de la siguiente ecuación:

$$F1\text{-}Score = \frac{Precision \bullet Recall}{Precision + Recall}$$

  Además, el añadir más datos no cambia el resultado de manera efectiva. Así mismo, cuando los falsos positivos y los falsos negativos son igualmente costosos, ambos impactaran al modelo casi de la misma manera.

- ***Matriz de confusión:*** proporciona una visión detallada de los resultados, obteniendo los valores TP, TN, FP y FN y el rendimiento para cada clase del modelo de manera detallada y mostrando cuando el modelo acierta en sus predicciones, así como sus desaciertos o fallos de predicción.

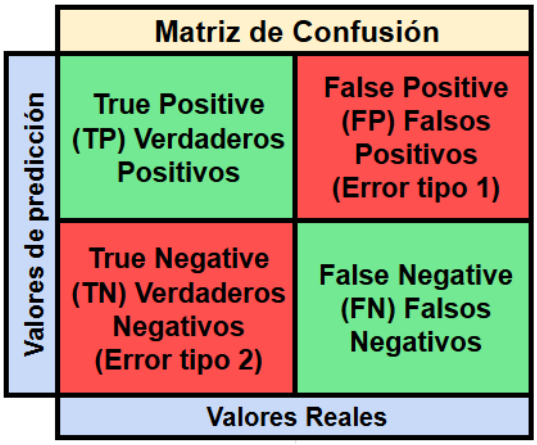

Figura 3.6. Matriz de confusión binaria, valores generados (TP, TN, FP y FN). Fuente: Imagen de elaboración propia

- ***AUC-ROC:*** La curva ROC (Curva de característica operativa del receptor) es una representación visual del rendimiento en todos los umbrales. AUC (Área bajo la curva) representa la probabilidad de que un modelo ante un par de valores positivos y negativos elegidos al azar califique al positivo más alto que al negativo. Es decir, Indica qué tan bien separa clases positivas de negativas. Cuanto más cerca de 1, mejor.

En lo que respecta a la ***detección de objetos***, las métricas más usadas son las siguientes:

- ***mAP (mean Average Precision):*** extiende el proceso de AP (Precisión promedio), calculado los valores de AP promedio para múltiples clases de objetos. Esta métrica es particularmente útil en detección de objetos con clases múltiples, proporcionando una

evaluación integral del rendimiento del modelo. El mAp al ser un promedio, se puede expresar a través de la siguiente expresión matemática.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i$$

donde,

- **N:** representa la cantidad de objetos.
- **AP:** representa la precisión promedio, la cual se puede expresar a través de la siguiente ecuación

$$AP = \frac{1}{N}\sum_{k=1}^{N} P(k)\Delta_r(k)$$

donde,

- $\boldsymbol{P(k)}$ Precisión en el punto/umbral k.
- $\boldsymbol{\Delta_r(k)}$ Diferencia respecto al rendimiento entre k y $k-1$.

- ***IoU:*** Intersección sobre la unión (*Intersection over Union*), métrica que cuantifica la intersección entre un bounding box predicho y uno de referencia (ground truth). Es decir, mide el grado de solapamiento o superposición entre la caja de entrenamiento y la de referencia, siendo particularmente relevante durante la evaluación de la precisión de la localización de objetos, la cual se define a través de la siguiente proporción:

$$IoU = \frac{\textit{área de intersección}}{\textit{área de unión}}$$

- ***Precisión y Recall por clase respeto a un umbral de IoU:*** procede de manera similar que, en clasificación, pero considerando si las cajas tienen suficiente IoU (mayor a 0.5) para contar como verdaderas positivas.

El nivel de confianza de las métricas mencionadas anteriormente, dependen además de la calidad del modelo, de los criterios implementados durante la evaluación y el tipo de tarea, por lo que no es raro encontrar que dos métricas tengan desempeños o niveles de confianza similares, así como también puede ocurrir que una de ellas se puede ver más favorecida que otra, bajo ciertas circunstancias, como lo es por ejemplo, el caso de la relación inversa entre precisión y recall.

Los criterios escogidos, afectan el orden en el que las métricas se ubican en una tabla comparativa, afectando directamente al algoritmo de detección de objetos, así como también puede ocurrir que, dependiendo de la implementación, cada una sea considerada como una especie de desempate. En general, se puede afirmar que no existe un consenso general en lo respecta a las herramientas de evaluación en este tipo de proyectos de investigación [22].

Posteriormente al uso de las métricas de rendimiento del modelo, es recomendable medir la calibración de este mediante el ECE, para comprobar la capacidad de interpretación y el grado de calibración del modelo, para de esta manera determinar si tras el entrenamiento adicional, se alteraron positiva o negativamente los valores del backbone.

A continuación, se valida la generalización y la robustez del backbone a través de la detección de datos fuera de la distribución (***Out-of-Distribution Detection ODD***), o con datos con ruido o anomalías. Luego, de ser posible, se debe evaluar la posibilidad de optimizar el modelo y, en caso de ser necesario, se debe considerar la optimización de los hiperparámetros (también conocidos como metaparámetros) utilizados durante el entrenamiento adicional.

Finalmente, para completar el proceso y tras tener la certeza de que el modelo se comporta de la manera esperada o mejor de lo esperado, se documentan todos los datos obtenidos a partir del finetuning y se guarda el modelo para su posterior uso o aplicación en tareas de detección de objetos, con la finalidad de encontrar los patrones o anomalías deseados en función de verificar la efectividad del modelo.

La optimización de los modelos de deep learning, independientemente de la tarea para la cual fueron desarrollados y entrenados, es un factor importante que considerar tras la culminación del entrenamiento y la fase de pruebas, ya que permite expandir y mejorar la utilidad del modelo, así como los fines para los cuales se puede implementar el mismo en un futuro, como lo puede ser el desarrollo de aplicaciones especializadas y otros sistemas que pueden ser de utilidad en diferentes campos e incluso a nivel social, dependiendo de la implementación del mismo.

Capítulo 4. DESARROLLO DEL PROYECTO

4.1 Planificación del proyecto

El presente Trabajo de Fin de Máster tiene como enfoque la investigación aplicada, combinando técnicas de aprendizaje profundo (*Deep Learning*) orientadas al área de la visión artificial (Computer Vision) y a la resolución de tareas relacionadas con el procesamiento digital de imágenes, pero enfocándose particularmente en la detección de objetos (*Object Detection*) con la finalidad de desarrollar modelos predictivos capaces de identificar factores y patrones de comportamiento en imágenes preprocesadas de animales salvajes y domésticos, que puedan estar asociados a enfermedades o condiciones anómalas, o inclusive a algún factor que este influenciando negativamente el entorno o habitad natural de dichas criaturas, lo cual además permitir el desarrollo de planes o acciones preventivas en pro del bienestar de dichas criaturas. Dicha tarea es posible de realizar a través del uso de redes neuronales con arquitecturas Backbone adaptadas a tareas de detección de objetos en imágenes.

A continuación, la metodología propuesta contempla un enfoque riguroso, sistemático y detallado en cada una de sus fases, con la finalidad de garantizar la calidad de los datos procesados, la robustez del modelo a desarrollar y la validez de los resultados obtenidos al final del proceso descrito a continuación.

Dicho esto, se dividirá la metodología del presente proyecto en varias fases secuenciales, diseñadas con la finalidad de obtener los mejores resultados posibles teniendo en consideración factores como el tiempo y la reducción del costo computacional en la medida de que sea posible, en función del equipo y las capacidades técnicas del mismo para la realización de cada fase.

4.1.1. Fase 1: Recolección y análisis de datos.

La primera fase es la de ***"Recolección y análisis de datos"***. Esta fase inicial se centrará en todo el proceso de selección, inspección y preparación de los datos que servirán de base tanto para el análisis como para la realización de todo el posterior trabajo experimental. Se considera el uso de variados datasets amplios y de código abierto disponibles en plataformas como Hugging Face, la cual permite acceder a colecciones de imágenes previamente curadas, etiquetadas y anotadas por comunidades científicas o proyectos institucionales, los cuales incluso han sido utilizados por empresas como Meta (Facebook, Instagram, etc) en proyectos de visión artificial entre otros, que a lo largo de los años han permitido el desarrollo y la mejora de redes neuronales como ConvNeXt y sus variaciones, así como de otros modelos de redes neuronales para la realización de tareas específicas.

En esta primera fase, el análisis preliminar incluye la verificación del tamaño y la resolución de las imágenes, el número de clases disponibles (especies, razas, enfermedades o condiciones), la calidad y consistencia de las etiquetas, así como el balance entre las diferentes clases. Además, se evaluará la naturaleza de las anotaciones del dataset (Es decir, si son cajas delimitadoras,

máscaras, puntos clave, etc.), lo cual permitirá determinar el tipo de arquitectura de red más conveniente a emplear más adelante.

Por otra parte, también se revisarán los aspectos éticos y legales del uso del dataset, verificando licencias, atribuciones y condiciones de uso para garantizar el cumplimiento normativo. Finalmente, se seleccionará el dataset (o posible combinación de datasets) que mejor se adapte a los objetivos del proyecto.

Antes de avanzar a la segunda fase, es importante tener en consideración la posibilidad de usar más de un dataset para poder lograr los objetivos planteados, dado que es importante verificar si dichos conjuntos de datos se adecuan a las necesidades del proyecto, así como a la tarea de detección de objetos, en caso contrario será necesario preparar al menos un dataset con las características faltantes para la identificación de patrones que puedan estar relacionados con enfermedades, agregando las anotaciones (bounding boxes) y etiquetas adecuadas para la clasificación y detección de objetos.

4.1.2. Fase 2: Preprocesamiento y aumentación de los datos

La siguiente fase contiene todo lo referente al ***"Preprocesamiento y aumentación de los datos"***, dónde una vez seleccionado el dataset, el primer paso a realizar es el preprocesamiento para transformar los datos en un formato más adecuado y optimizado para ser interpretado por los modelos de aprendizaje automático a desarrollar. Dicho preprocesamiento constará de los siguientes pasos:

- ***Redimensionamiento*** de imágenes para unificar las dimensiones de entrada de estas. Lo que es particularmente beneficioso al trabajar con distintos datasets, ya estos pueden contener imágenes ya redimensionadas o no, o puede haber diferencias de dimensión entre las imágenes entre datasets.
- ***Normalización*** de valores de píxel, lo cual consiste en un re-escalado de los datos de las imágenes (pixeles) para pasarlos de valores que se encuentren en el rango entre 0 y 255, a valores más pequeños en un rango entre 0 y 1.
- ***Conversión de etiquetas*** a formatos estándar acorde al framework a implementar (que en este caso se tratará de pytorch), ya que es necesario para los pasos posteriores que las etiquetas estén en un formato estructurado como diccionarios por imagen, similar al que se muestra a continuación:

```
{
    "boxes": Tensor[N, 4], # [xmin, ymin, xmax, ymax]
    "labels": Tensor[N],   # clase (entero)
    "image_id": Tensor[1],
    "area": Tensor[N],     # area de cada caja
    "iscrowd": Tensor[N]  # 0 o 1
}
```

- ***Eliminación de imágenes corruptas o inconsistentes*** mediante validaciones automatizadas (lo cual se puede hacer mediante pytorch que permite automatizar dentro del dataset), o con validaciones concretas antes de realizar la carga de los datos.

Una vez culminado todo el preprocesamiento de los datos seleccionados. Posteriormente, se aplican técnicas de aumentación de datos que permiten incrementar artificialmente el volumen y la diversidad del conjunto de entrenamiento. Este paso es crucial para evitar problemas como el sobreajuste de los datos y para aumentar la capacidad del modelo para generalizar ante nuevas situaciones de visión artificial. Algunas de las transformaciones a aplicar incluyen:

- Rotaciones aleatorias, recortes, traslaciones y escalados.
- Cambios de iluminación, contraste y saturación.
- Aplicación de ruido, desenfoques y transformaciones geométricas.

Estas operaciones se pretenden implementar mediante bibliotecas especializadas como *Albumentations*, la cual permite definir pipelines de transformación eficientes y fáciles de reproducir.

4.1.3. Fase 3: Selección, implementación y entrenamiento del modelo predictivo.

La tercera fase consta de todo el proceso de ***"Selección, implementación y entrenamiento del modelo predictivo"***. En esta fase, se elige una red neuronal pre-entrenada (o varias para fines comparativos posteriores de rendimiento) que actuará como Backbone. Estas redes han sido previamente entrenadas sobre grandes corpus de imágenes (como ImageNet), lo que les permite aprender representaciones generales útiles para tareas diversas. Algunos ejemplos relevantes incluyen:

- **ResNet:** son las redes neuronales convolucionales básicas más conocidas y usadas por la comunidad con conexiones de salto y normas de lote, la cual es excelente para tareas de clasificación profunda y detección de objetos. Particularmente ResNet-50 a pesar de ser un modelo de clasificación de imágenes, puede usarse como backbone o extractor de características para tareas de detección de objetos, por lo cual se tomará en consideración para comparar su rendimiento con los otros modelos propuestos.
- **ConvNeXt:** es una red puramente convolucional con numerosas mejoras arquitectónicas modernas que incluyen convoluciones en profundidad, bloques de cuello de botella invertidos, grandes núcleos convolucionales y una mayor proporción de capas asignadas a la tercera etapa. Es un modelo ConvNet puro que aprovecha el poder de las convoluciones separadas por profundidad para mejorar el rendimiento en tareas de computer vision.

  Esta arquitectura mejora el rendimiento de las arquitecturas convolucionales a escala, al tiempo que mantiene sus potentes capacidades de detección y segmentación de objetos. ConvNeXt forma parte de una familia de modelos que incluyen los tipos: Tiny, Small, Base, Large y XLarge, cada uno diseñado para casos de uso específicos, lo que le permite abarcar varias tareas de computer vision.

Se toman en especial consideración ConvNeXt-Tiny y ConvNeXt-Base, para su implementación en el presente proyecto.

- **Swin Transformer:** es una arquitectura de transformadores que incorpora representaciones jerárquicas, invariancia de traslación, y mayor localidad y eficiencia en los ViTs al realizar la atención únicamente dentro de ventanas espaciales y fusionar estas ventanas iterativamente.

  En resumen, es una arquitectura basada en atención, eficaz para estructuras visuales complejas.

La red Backbone se incorpora dentro de una arquitectura de detección de objetos (por ejemplo, Faster R-CNN, YOLOv5/u8, RetinaNet), las cuales se dividen en dos categorías:

- **Detectores de dos etapas:** entre los que se encuentran Faster R-CNN y Mask R-CNN que se encargan primero de localizar y luego de clasificar los objetos de las imágenes en regiones separadas, ofreciendo alta precisión, pero menor velocidad de procesamiento.
- **Detectores de una sola etapa:** como YOLO, SSD y RetinaNet, los cuales realizan ambas tareas de manera simultánea para lograr una mayor velocidad de procesamiento, lo cual es ideal para aplicaciones en tiempo real.

Luego se procede a configurar el entorno de entrenamiento utilizando PyTorch y definiendo:

- Las funciones de pérdida (por ejemplo, focal loss, smooth L1).
- El optimizador (como Adam, AdamW o SGD).
- La tasa de aprendizaje (Learning Rate o lr) y el número de épocas procurando que sea una cantidad adecuada para evitar el overfitting.
- Las métricas de evaluación durante y después del entrenamiento.

Posteriormente, se incorporará una estrategia de entrenamiento progresivo, dónde Inicialmente ***se congelarán los pesos del backbone preentrenado***, lo que permitirá ***entrenar únicamente las cabezas de la red***, acelerando en consecuencia el entrenamiento y reduciendo el riesgo de sobreajuste en las primeras etapas. Esta técnica permite adaptar modelos robustos a tareas especializadas con menor cantidad de datos.

Una vez estabilizado el entrenamiento de la cabeza, se considerará el descongelamiento parcial o total de las capas profundas del backbone para realizar el fine-tuning global del modelo, así como el entrenar una segunda cabeza para la detección de los patrones anómalos, dejando la primera cabeza entrenada para identificar el animal en la imagen. Este proceso será acompañado por técnicas de control como Early Stopping, reducción adaptativa del learning rate (scheduler) y ***monitoreo continuo***, mediante visualización de métricas clave como lo son: la pérdida (***loss-rate***), la precisión (***precision***), el mAP (***mean average precision***), entre otras.

Por lo tanto, se implementarán registros del rendimiento (logs) y visualizaciones de las curvas de pérdida y evolución de métricas para así poder identificar los momentos óptimos de parada, así como el verificar la convergencia del modelo y prevenir el sobre entrenamiento (Overfitting).

4.1.4. Fase 4: Evaluación y análisis de resultados.

La siguiente fase consiste en el proceso de "Evaluación y análisis de resultados", la cual se implementará tras finalizar el entrenamiento del modelo, procediendo a evaluarlo sobre un conjunto de prueba independiente. Dicha evaluación tiene como finalidad el medir la capacidad del modelo para detectar objetos correctamente en imágenes nunca vistas, en función del objetivo general del presente proyecto.

Para llevar a cabo la evaluación antes mencionada, se emplearán las siguientes métricas:

- **Precisión (precision):** esta métrica representa el porcentaje de predicciones correctas sobre el total de predicciones.
- **Recall:** esta métrica representa proporción de objetos correctamente detectados respecto al total real.
- **F1-score:** esta métrica representa la media armónica entre precisión y recall, útil en contextos con clases desbalanceadas.
- **mAP (mean Average Precision):** esta es una métrica estándar usada en detección de objetos que resume el rendimiento global en todas las clases.
- **Matriz de confusión:** herramienta que permite visualizar el desempeño de un algoritmo de aprendizaje supervisado, que permite apreciar qué tipos de aciertos y errores está teniendo el modelo a la hora de pasar por el proceso de aprendizaje con los datos.

Finalmente, se analizarán errores sistemáticos, falsas alarmas y omisiones, a través de la generación de matrices de confusión, curvas PR y visualizaciones de detecciones correctas y erróneas. En paralelo, se explorará la correlación entre los patrones detectados y la aparición de signos visuales asociados a comportamientos anómalos o posibles factores relacionados con enfermedades en animales.

Este análisis puede realizarse de forma cuantitativa y también cualitativa, mediante la comprobación e inspección visual por expertos (como veterinarios, etólogos, ecólogos, biólogos, etc.), así como en documentación científica y médica relacionada, lo que permitirá corroborar la eficacia y la capacidad de acierto del modelo.

4.1.5. Fase 5: Ajustes y documentación de resultados.

La fase final consiste en ***"Ajustes y documentación"***. Durante esta etapa se realizarán todos los ajustes y consideraciones necesarios tras el periodo de pruebas, para finalmente documentar y presentar los resultados obtenidos en el presente proyecto, en vista de su implementación practica a futuro. También se pretende documentar los distintos problemas y complicaciones encontrados durante cada fase de la realización del presente trabajo de fin de grado, así como aquellos detalles o características adicionales presentes en datasets, modelos, entre otros, que terminaron por aportar resultados favorables o facilidades durante la fase de entrenamiento y pruebas realizadas para determinar la eficacia y el desempeño del modelo desarrollado.

Capítulo 5. RESULTADOS

Durante la fase de experimentación de realizaron pruebas con varios datasets y modelos obtenidos de la web Hugging Face principalmente, los cuales al estar preentrenados poseen una estructura dividida en tres subconjuntos: train, validation y test. Sin embargo, también se realizaron pruebas con datasets sin el conjunto validation.

5.1 Dataset Francesco/animals-ij5d2.

Conjunto de imágenes [5] preentrenado de Hugging Face y orientado a la detección de objetos, como se puede apreciar en el siguiente bloque a través su estructura y en las características que lo componen, particularmente en el campo features donde la característica "objects" suele contener las anotaciones relacionadas con las bounding boxes utilizadas para crear los recuadros que delimitaran los objetos de interés.

```
DatasetDict({
  train: Dataset({
    features:['image_id','image','width','height','objects'],
    num_rows:700
  })
  validation: Dataset({
    features:['image_id','image','width','height','objects'],
    num_rows:100
  })
  test: Dataset({
    features:['image_id','image','width','height','objects'],
    num_rows:200
  })
})
```

La característica "objects" suele contener información bastante detallada como:

- **Clases:** representan las categorías de los objetos presentes en la imagen, como lo son el tipo de animal, objeto, enfermedad, acción, etc.
- **Bounding Boxes:** representan las coordenadas de las cajas delimitadoras, las cuales indican tanto la ubicación como el tamaño de cada objeto.

Si el dataset estuviera orientado a la clasificación de imágenes, no necesitaría la caracteristica "objects", aunque las demás si pudieran estar presentes ya que ayudan a describir la imagen como tal. Sin embargo, también se puede encontrar datasets de clasificación que el campo "features" solo continen las características "imagez "labels". Este dataset cuenta con 11 clases definidas, las cuales son:

['animals', 'cat', 'chicken', 'cow', 'dog', 'fox', 'goat', 'horse', 'person', 'racoon', 'skunk']

Retomando el caso de estudio, se tiene que el subconjunto "train" es el que se usara para comprobar que el preetrenamiento de clasificación funcione de manera adecuada tanto para clasificación como para detección de objetos, puesto que se requiere de ambos para la correcta determinación de enfermedades y patrones anómalos en animales en pasos futuros.

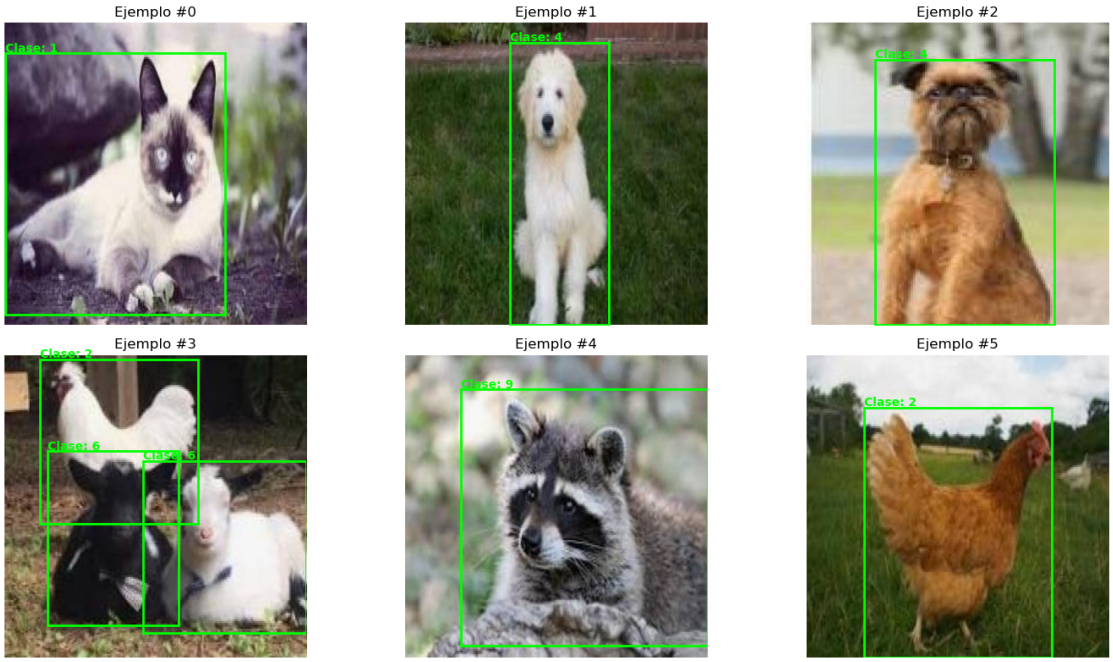

Figura 5.1. Imagen resultante del dataset Francesco/animals-ij5d2 de Hugging Face, con bounding box etiquetado por clase, obtenidos de los features del subconjunto train. Fuente: elaboración propia.

Como se puede apreciar en la Fig 5.1, el dataset es capaz de etiquetar una imagen haciendo uso de las clases predefinidas del 0 al 10, con la identificación correspondiente por tipo (animals, cat, chicken, cow, dog, fox, goat, horse, person, racoon, shunk), así mismo es capaz de generar e identificar varios animales diferentes en una imagen, generando bounding boxes (cajas limitadoras) para cada uno con su respectiva etiqueta, permitiendo agregar títulos adicionales por imagen generados manualmente (Ejemplo # 1, etc). Cabe destacar que dichos títulos no pertenecen a las características de etiquetado predefinidas.

El resultado obtenido en la prueba anterior sirve para corroborar que el dataset seleccionado funciona correctamente, ya que es capaz de clasificar y de detectar objetos de la manera esperada. Por lo que se puede comenzar a probar con los modelos de backbones seleccionados, el ConvNeXt-B entrenado en IN-21k para datasets enormes y equipos con altos recursos y el ConvNeXt (tiny-sized model) que no demanda una excesiva cantidad de recursos y tiene un buen desempeño en general.

5.2 Resultados para ConvNeXt-Base para clasificación:

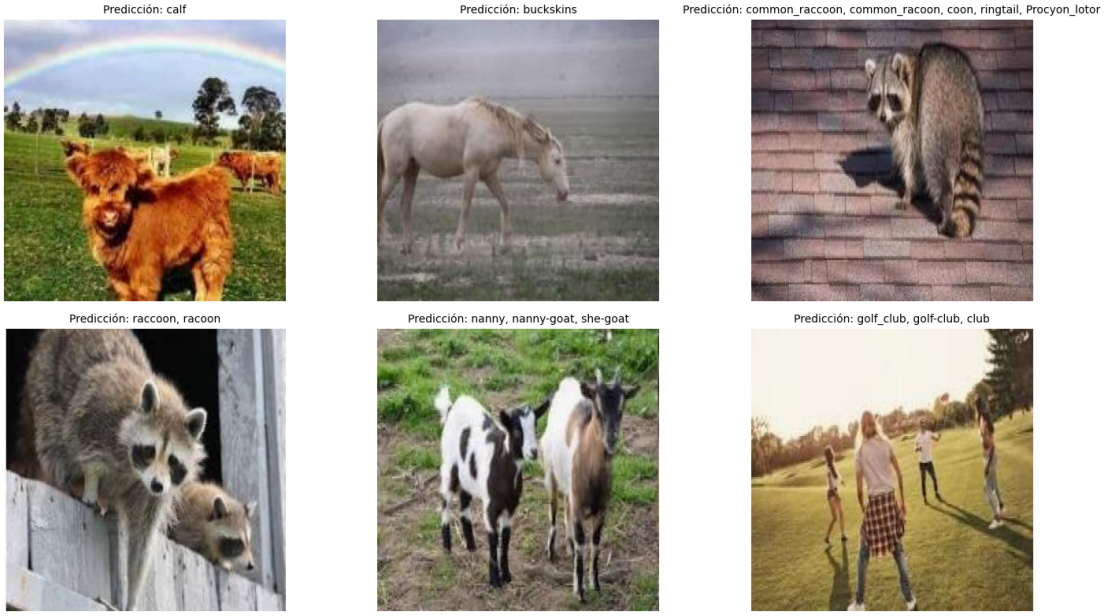

Figura 5.2. Imágenes del dataset Francesco/animals-ij5d2 de Hugging Face, con etiquetado por tipo, para clasificación usando el modelo de clasificación pre-entrenado ConvNeXt-Base. Fuente: elaboración propia

Como se puede apreciar, el backbone muestra un buen desempeño al clasificar las imágenes, a excepción de la segunda imagen que usa el término "Buckskins" el cual en ingles hace referencia tanto a la ropa hecha de piel de animal, así como para describir el color del pelaje de un caballo, lo cual no es una mala clasificación, pero es un poco ambigua. Mientras que la última imagen se centra en el fondo y no en las personas, lo cual no es preocupante porque en tareas de clasificación se obtiene una etiqueta general por imagen, por lo que para resultados más específicos es necesario dar un paso más y pasar a la detección de objetos, lo cual se llevara a cabo en pasos posteriores.

En conclusión, en lo que respecta a la clasificación, ConvNeXt-Base genera buenos resultados de etiquetado.

5.3 Resultados para ConvNeXt-tiny para clasificación:

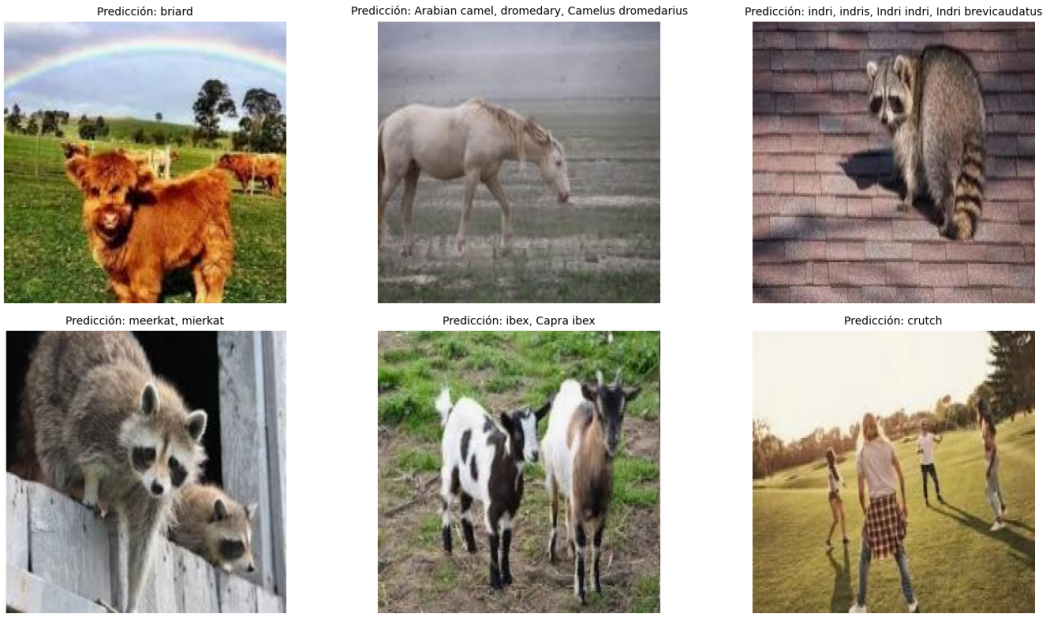

Figura 5.3. Imágenes del dataset Francesco/animals-ij5d2 de Hugging Face, con etiquetado por tipo, para clasificación usando el modelo de clasificación pre-entrenado ConvNeXt-Tiny. Fuente: Vs Code, elaboración propia.

Sin embargo, ConvNeXt-tiny a pesar de ser más veloz y eficiente, muestra fallos considerables de etiquetado para la mayoría de las imágenes, en lo que respecta a tareas de clasificación. Sin embargo, antes de pensar en descartar este modelo, es conveniente valorar su desempeño en tareas de detección de objetos, ya que podría el permitir optimizar las detecciones por cabeza o, por el contrario, puede no aportar cambios significativos respecto a ConvNeXt-Base.

En conclusión, ConvNeXt-tiny no genera buenos resultados de etiquetado para tareas de clasificación, respecto al dataset seleccionado para las pruebas.

5.4 Pruebas fallidas para ConvNeXt-Tiny y para ConvNeXt-B para detección:

En lo que respecta a las pruebas de **ConvNeXt-Tiny (facebook/convnext-tiny-224) y ConvNeXt-B (facebook/convnext-base-224-22k)** para tareas de detección de objetos, surgieron complicaciones relacionadas con la capacidad y los recursos de procesamiento del equipo personal empleado para dichas pruebas. El problema proviene de tratar de usar cualquiera de los dos modelos anteriores como backbones para detección de objetos sobre el dataset antes mencionado, ya que durante el procesamiento de los datos se saturan los recursos del equipo empleado para realizar cada tarea de cómputo requerida para el entrenamiento del modelo.

La segunda parte de estas pruebas se realizaron en un equipo con mayor cantidad de recursos de procesamientos y a nivel gráfico, pero los resultados obtenidos fueron similares en el aspecto de que antes de que siquiera pudiera terminar el proceso “el kernel moría por falta de memoria", acorde al mensaje de error obtenido. De momento, las principales diferencias entre ambos

equipos, además del aumento de temperatura a nivel de hardware de estos, se encuentra en la cantidad de tiempo transcurrido para obtener el mensaje de error, o incluso reinicios del sistema en el caso del equipo con menor capacidad de procesamiento.

5.5 Definición del modelo con cabeza de detección simplificada.

Una manera de evitar la saturación de recursos es la de entrenar el modelo por cabeza (capa), de esta manera se puede definir el modelo con cabezas simplificadas entrenadas para clasificación o detección, incluso se puede implementar para entrenar varias cabezas para detecciones específicas. Tras definir la cabeza o capa simplificada, se puede proceder al entrenamiento de esta congelando el backbone, lo que detiene el entrenamiento de los pesos de este, para enfocarse solo en el entrenamiento de esta capa personalizada, lo cual permite usar un backbone de clasificación como extractor de características.

La desventaja de este proceso es que, al estar usando un modelo pre-entrenado, puede ocurrir que la capacidad de aprender nuevas características del nuevo dataset (en este caso el seleccionado para las pruebas) en especial si dicho dataset presenta diferencias significativas con el dataset usado para el entrenamiento previo.

La idea en este caso es la de moldear a este extractor de características para que pueda convertir dichas características en valores o clases (como los implementados en el proceso de clasificación), así como el asignar valores a las bounding boxes o cajas delimitadoras de la forma [x, y, w, h].

Luego se optimiza el modelo anterior, lo cual consiste en entrenar solo los parámetros que, si se pueden actualizar, que en este caso son los de la cabeza definida anteriormente. Como paso adicional, se recomienda el transformar el dataset a uno compatible con Pytorch (es decir, transformar todos los parámetros y características a tensores) para poder realizar el entrenamiento.

Tras las preparaciones previas, se extrae el siguiente ejemplo antes del entrenamiento del modelo:

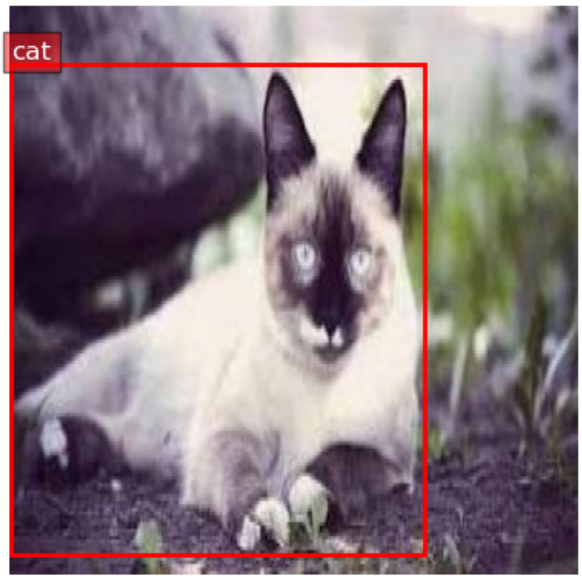

Figura 5.4. Imagen del dataset Francesco/animals-ij5d2 de Hugging Face, con etiquetado y detección por tipo, a través de bounding box, generado a partir de un modelo simplificado y congelado de ConvNeXt-Tiny con una sola capa optimizada para entrenamiento. Fuente: elaboración propia.

Otra estrategia por probar que evitaría el limitar la capacidad de aprendizaje del modelo, implica el aplicar un fine-tuning que consista en un descongelamiento parcial del backbone, descongelando las ultimas capas para ajustarlas a la tarea (clasificación o detección según sea el caso). El siguiente paso consistirá en evitar el overfitting del entrenamiento agregando la caracteristica nn.Dropout a la cabeza o capa de clasificación, luego se reducirá la tasa de aprendizaje a 1e-5 debido a que suele ser el estándar utilizado en backbones pre-entrenados. Finalmente dependiendo de los resultados obtenidos, se puede probar con diferentes tasas de aprendizaje implementando un ajuste dinámico de dicha tasa durante el entrenamiento a través de un **learning rate scheduler**.

5.6 Otros aspectos por considerar.

Dado que las pruebas para detección de objetos fallaron y considerando los resultados antes obtenidos para clasificación, conviene el considerar no solo hacer uso de otro dataset público orientado a tareas de clasificación, sino también el adaptar el dataset actual para que se enfoque en tareas de clasificación. Luego solo queda el configurar adecuadamente la cantidad de épocas o iteraciones adecuada para obtener el mejor modelo. En el caso de ConvNeXt-tiny con 50 iteraciones se obtienen los siguientes resultados:

- **Resultados variando la tasa de aprendizaje:**
  - **Peor resultado:** Epoch 1/50 | LR: 2.0e-06 | Train Loss: 2.4978 | Val Acc: 21.00 %
  - **Mejor resultado:** Epoch 39/50 | LR: 1.4e-06 | Train Loss: 0.3602 | Val Acc: 90.00 %
- Resultados sin variar la tasa de aprendizaje:
  - **Peor resultado:** Epoch 2/50 | Train Loss: 0.1319 | Val Acc: 87.00 %
  - **Mejor resultado:** Epoch 23/50 | Train Loss: 0.0010 | Val Acc: 91.00 %

En el caso de ConvNeXt-base con 50 iteraciones se obtienen los siguientes resultados:

- **Resultados variando la tasa de aprendizaje:**
  - **Peor resultado:** Epoch 1/50 | LR: 2.0e-06 | Train Loss: 2.4157 | Val Acc: 6.00 %
  - **Mejor resultado:** Epoch 22/50 | LR: 6.9e-06 | Train Loss: 2.0938 | Val Acc: 27.00 %
- **Resultados sin variar la tasa de aprendizaje:**
  - **Peor resultado:** Epoch 1/50 | Train Loss: 2.3267 | Val Acc: 16.00 %
  - **Mejor resultado:** Epoch 47/50 | Train Loss: 1.0299 | Val Acc: 39.00 %

En ambos casos se definieron 50 iteraciones ya que a partir de ese punto los modelos mostraron señales de no continuar mejorando, por lo que el early stopping configurado con el parámetro $patience = 20$ para las pruebas con 150 iteraciones, así como el $patience = 10$ para las pruebas con 50 iteraciones configuradas, se activaron deteniendo el modelo para evitar el overfitting.

Tras finalizar el entrenamiento, se guardan los modelos con los mejores resultados en clasificación, para implementarlos en la siguiente fase de pruebas, la cual requiere de:

- Determinar 3 subtareas de etiquetado y detección.
- Definir y simplificar 3 cabezas del modelo, una para cada subtarea, congelando el resto del modelo.
- Dichas cabezas o capas se encargarán de:
  - **Cabeza 1:** salida softmax que determine el tipo de animal en la imagen.
  - **Cabeza 2:** salida para el bounding box que denote la enfermedad.
  - **Cabeza 3:** salida softmax que determine el tipo de enfermedad.

No obstante, actualmente debido a la falta de datasets disponibles que permitan denotar las enfermedades en animales a través bounding boxes, fue necesario el preparar de manera manual un dataset solo para este fin, tomando una muestra del dataset público de Kaggle dogs-skin-diseases-image-dataset [6] redimensionándolo, rescaldándolo, normalizándolo, agregando imagenes de dominio público para incluir gatos con las mismas enfermedades a la muestra , para finalmente preparar su estructura para el entrenamiento con los modelos antes mencionados, o en su defecto de otras redes convolucionales que se adapten mejor a esta tarea de detección como lo es el caso de YOLO.

La consideración de incorporar la red convolucional YOLO para el entrenamiento de detección debido al problema antes expuestos para encontrar datasets de enfermedades de animales preentrenados para la detección de objetos, ocasionó que fuera necesario cambiar el enfoque del entrenamiento de las tres cabezas de detección, a otro enfoque donde se pretende mantener el modelo de convNeXt entrenado para clasificación del tipo de anima y aparte entrenar un modelo con YOLO para la detección de enfermedades y su delimitación en bounding boxes.

Este nuevo enfoque implica el entrenar cada modelo para una tarea (clasificación o detección) y luego usar los modelos con mejores resultados de ambas redes neuronales convolucionales para luego unir ambos resultados en un solo pipeline secuencial, donde la salida del primer modelo (Detección de objetos con YOLO) se convertirá en la entrada del segundo modelo (ConvNeXt para clasificación), convirtiendo el proceso en una tarea de procesamiento secuencial.

La inferencia en cascada [7] no es el mejor enfoque para mejorar la precisión general, pero si es bastante útil para casos como este ya que se enfoca en mejorar el rendimiento general. Si el caso fuera diferente y se pretendiera usar dos o más modelos para realizar la misma tarea y luego comparar resultados, hay estrategias más eficientes como lo son el stacking (combina la fuerza predictiva de ambos modelos maximizando la precisión), el bagging (reduce varianza y mejora la estabilidad en modelos susceptibles al overfitting) y el boosting (reduce el sesgo y mejora la precisión).

5.7 Adaptación del dataset de enfermedades para YOLO.

Dado que el dataset de enfermedades no contaba con las anotaciones necesarias para el entrenamiento de detección, fue necesario la implementación de la versión web de label-studio [21] para la generación de las bounding boxes y sus respectivas anotaciones en formato YOLO.

Label-studio va más allá del diseño de las cajas delimitadoras y de su representación tensorial, también genera un archivo de importación con parte de la estructura adecuada para su uso en la red convolucional Yolo, separado en carpetas las imágenes (images/) y las etiquetas (labels/), generando un txt llamado "classes" que contiene los nombres de cada clase o categoria que representa a cada enfermedad, así como también genera un archivo json denominado "notes".

Esto es de especial, utilidad ya que adiconalemente se debe generar un archivo yaml que contenga estos datos, asi como la ruta local del dataset que contiene las enfermedades junto con las rutas de las carpetas train, test y val que se deben crear dentro de la carpeta images/ y en labels/. Donde se distribuyeron los datos en proporciones de 80 % train, 10 % test y 10 % val usando las 400 imágenes y anotaciones del conjunto.

Es importante aclarar que solo 100 de las 400 imágenes no poseen anotaciones asociadas en labels/, ya que estas 100 imágenes representan el estado "Healthy" y solo contienen imágenes de perros y gatos sanos. Estas imágenes también se encuentran distribuidas entre las carpetas train, test y val en las mismas proporciones de 80 %, 10 % y 10 % respectivamente.

5.8 Entrenamiento con YOLO (You Only Look Once).

El modelo Yolo11 [2] contiene diferentes modelos adaptados a diferentes necesidades, los cuales se clasifican como: nano, small, medium, large y extra-large, de entre los cuales se realizaron pruebas con las versiones nano, small y medium debido a los recursos de procesamiento disponibles, las cuales se denominan como yolo11n, yolo11s y yolo11m respectivamente, todos preentrenados con imagenes de dimensiones 640x640px, lo que en consecuencia define el tamaño de redimensionamiento de las imágenes del dataset de enfermedades a 640x640 para adaptarse lo más posible al modelo.

Es importante resaltar que aun con el muestreo y la reconstrucción manual del dataset, la calidad de una buena porción de las imágenes no es buena, por lo que los modelos entrenados tienen altas posibilidades de caer en overfitting durante el entrenamiento y filtrar todo solo a las imágenes de calidad decente, hubiera reducido considerablemente el dataset, lo cual para propositos de entrenamiento de modelos, no es adecuado.

5.8.1. Prueba con YOLOv11n (YOLOn).

			Box(P R mAP50 mAP50-95): 100% ———— 2/2 2.4it/s 0.8s1.9s			
Class	**Images**	**Instances**	**P**	**R**	**mAP50**	**mAP50-95**
all	40	48	0,897	0,413	0,508	0,345
demodicosis	10	13	1	0,302	0,493	0,299
fungal_infections	8	22	0,795	0,0909	0,126	0,0774
ringworm	12	13	0,897	0,846	0,907	0,657

Tabla 5.1. Tabla de resultados de entrenamiento para el modelo YOLO11n para 50 épocas/iteraciones sobre el dataset de enfermedades de la piel en perros y gatos (Dog_skin_disease_modify). Muestra las clases (enfermedades por separado y en conjunto), la cantidad de imágenes, las instancias (cantidad de cajas) y las métricas de rendimiento: Box Precision (P), Box Recall (R), mean average precision con umbral IoU de 0.50 y de 0.50 a 0.95 (mAP50 y mAP50-95 respectivamente). Fuente: Elaboración propia.

Estos resultados corresponden a las imágenes de la carpeta de validación "val", la cual posee 40 imágenes, distribuidas en 10 imágenes por cada enfermedad y otras 10 sin anotaciones que corresponden al estado "Healthy" o saludable, el cual sirve para comparar y probar la capacidad de detección del modelo.

Como se puede apreciar, entre las tres enfermedades listadas, solo el anillo de gusanos (ringworm) las métricas en general arrojan buenos resultado, en su mayoría cercanos a 1 a excepción de mAP50-95 que es de 0.657, es decir de un 65 % aproximadamente, lo que no representa un mal resultado, pero idealmente se esperan valores por encima del 70 %. Por otro lado, las otras dos enfermedades, demodicosis e infecciones fúngicas (hongos) muestran valores altos de precisión a costa del recall, que muestra resultados menores a 0.5 en general.

Variando la cantidad de épocas y aplicando técnicas de aumentación de datos (rotacion, saturación y zoom,), así como el weight_decay de 0.001 a 0.05, entre otros, la mejoría percibida del modelo es mínima, pero denotando una clara tendencia a la mejoría. El problema viene de los valores obtenidos para las infecciones fúngicas en el recall, ya que, al medir la capacidad del modelo de detectar todos los objetos reales en una imagen, el obtener valores tan bajos indica que está teniendo dificultades para detectar adecuadamente todas las cajas anotadas en cada imagen, mostrando una buena capacidad de acierto en algunos casos, pero baja capacidad de acierto en el resto de los casos, lo cual se debe en parte a la mala calidad de las imágenes de esa clase.

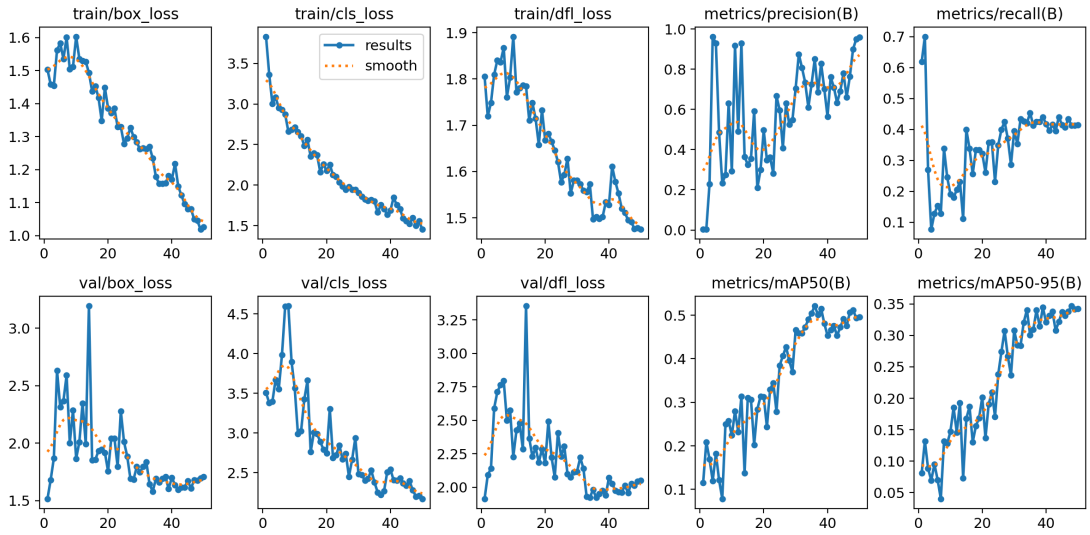

Figura 5.5. Comparación de métricas de rendimiento y de perdida para el entrenamiento (train) y para la validación (val) para 50 épocas/iteraciones. Fuente: elaboración propia.

Las gráficas de la Fig 5.5 reflejan la perdida (box_loss, cls_loss y dfl_loss) así como las métricas de rendimiento (precision, recall y mAP) tanto para el proceso de entrenamiento como para el de validación. En el caso de la perdida, los valores muestran tendencia a decrecer con poca fluctuación en el entrenamiento y con fluctuación moderada en la etapa de validación, mientras que en lo que respecta al rendimiento, la tendencia en general es de crecimiento. Estos son buenos indicadores para la etapa de entrenamiento ya que, a pesar de los valores obtenidos anteriormente, el modelo muestra que la perdida de localización (box_loss), de clasificación (cls_loss) y focal de distribución (dfl_loss) tienden a disminuir a mayor cantidad de iteraciones, mientras que en lo que respecta a la etapa de validación, donde se evalúa la capacidad de generalización del modelo para imágenes que no ha visto durante el entrenamiento, si bien la tendencia a disminuir también es positiva, si hay secciones donde se detiene o comienza a aumentar mientras en su contraparte en el entrenamiento sigue disminuyendo, puede considerarse una señal de sobre ajuste.

Por otro lado, en lo que respecta al rendimiento el umbral IoU (Intersection over Union) que mide el solapamiento entre la caja delimitadora predicha y la caja real, para el caso del mAP50 tener valores mayores a 0.50 normalmente se considera un acierto, ya que esta métrica indica el rendimiento de la detección de objetos con una localización relativamente flexible, por lo que en general se considera que el modelo acierta al tener un valor de 0.508 para el rendimiento general del modelo. No obstante, en lo que respecta a una métrica estándar y más estricta como lo es mAP50-95 que mide el rendimiento promediando el mAP calculado en 10 umbrales de IoU diferentes (de 0.50 a 0.95, en pasos de 0.05). Es decir, valores bajos en esta métrica denotan que el modelo tiene problemas para detectar y localizar los objetos.

Figura 5.6. Muestra de aciertos (verdaderos positivos) en la detección de las enfermedades de la piel en perros y gatos. Fuente: YOLO, elaboración propia.

La Fig 5.6 Muestra los aciertos en la detección correcta de las enfermedades de la piel en perros y gatos, delimitando e identificando adecuadamente (acorde a la clasificación original del dataset usado) las enfermedades, fallando solo en dos ocasionesen las imágenes 12 y 15, así como omitiendo las anotaciones en las imágenes donde no aparecen índices de enfermedad visible, ya que corresponden a los animales sanos del dataset.

Figura 5.7. Muestras de aciertos y desaciertos en la detección de enfermedades, con tres falsos negativos (imágenes 4, 7 y 8) en los que el modelo no detecto las enfermedades ringworm, fungal infection y demodicosis respectivamente. Fuente: YOLO, elaboración propia.

Como se puede apreciar en la Fig 5.7, cuando el modelo falla en la detección, no marca nada incluso en imágenes que claramente muestran enfermedades en la piel, así como en aquellas donde realmente el indicio de enfermedad es apenas distinguible como en el caso de la imagen 8 (última imagen de la rejilla).

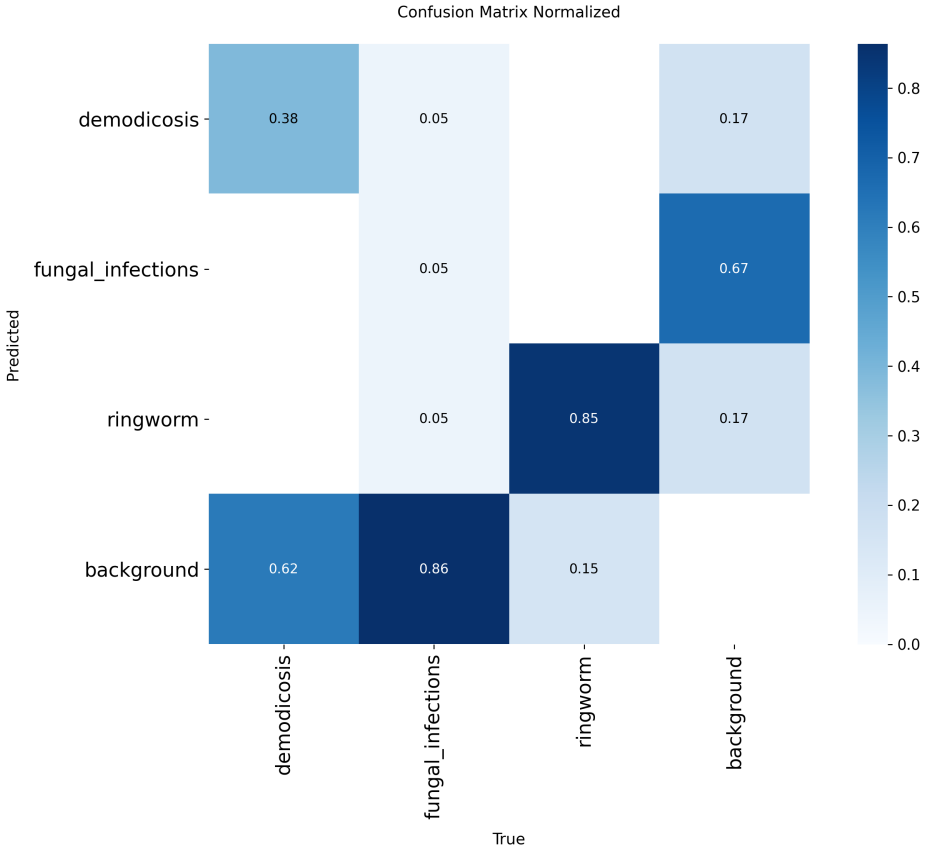

Figura 5.8. Matriz de confusión normalizada para yolo11n, mostrando las coincidencias y el porcentaje de acierto del modelo por clase, siendo background (denominación de yolo) el fondo de las imágenes del dataset. Vertical (valores predichos) vs horizontal (valores reales). Fuente: YOLO, elaboración propia.

Los resultados de la matriz de confusión muestran lo siguiente:

- **Demodicosis:** 62 % de las veces se confunde con background y 38 % de las veces acierta en la detección.
- **Fungal Infections (infecciones fúngicas):** 86 % se confunde con background, 5 % de las veces el modelo lo confunde con ringworm o demodicosis, para luego acertar solo el 5 % de las detecciones. Lo cual tiene sentido, ya que las imágenes de esta clase son las que tienen peor calidad entre las tres clases.
- **Ringworm (Anillo de gusanos):** 85 % de aciertos y solo 15 % de fallos de detección.

Dados los resultados anteriores, las siguientes pruebas fueron orientadas a mejorar la métrica mAP50-95 usando modelos con más características y especializados como yolo11s, cuyo costo computacional en tiempo fue de 15 a 21 minutos de ejecución a comparación de yolo11n que tomó de 2 a 5 minutos de ejecución por prueba. Así como también se realizaron pruebas con yolo11m el cual mostró mejorías respecto a los dos anteriores, pero no considerablemente diferentes además del tiempo de ejecución que aumentó significativamente a 4 horas para 50 iteraciones, hasta 12 horas continuas de ejecución para 150 iteraciones, ambas con early stopping fijado en 20 iteraciones de rango de mejora.

5.8.2. Prueba con YOLOv11m (YOLOm).

			Box(P R mAP50 mAP50-95): 100% ———— 2/2 1.8it/s 1.1s2.8s			
Class	**Images**	**Instances**	**P**	**R**	**mAP50**	**mAP50-95**
all	40	48	0,801	0,379	0,504	0,342
demodicosis	10	13	0,889	0,308	0,448	0,267
fungal_infections	8	22	0,604	0,136	0,14	0,086
ringworm	12	13	0,91	0,692	0,925	0,674

Tabla 5.2. Tabla de resultados de entrenamiento para el modelo yolo11m, para 150 épocas/iteraciones sobre el dataset de enfermedades de la piel en perros y gatos (Dog_skin_disease_modify). Muestra las clases (enfermedades por separado y en conjunto), la cantidad de imágenes, las instancias (cantidad de cajas) y las métricas de rendimiento: Box Precision (P), Box Recall (R), mean average precision con umbral IoU de 0.50 y de 0.50 a 0.95 (mAP50 y mAP50-95 respectivamente). Fuente: Elaboración propia.

Tras detenerse en la época 112 se obtienen los resultados de la tabla 5.2 y, como se puede apreciar en la tabla anterior, fungal_infections muestra una mejoría mínima en sus valores, a costa de la disminución mínima de los valores de las demás clases, en todas las métricas y aun así el mAP50-95 se mantiene prácticamente igual, lo cual muestra que las limitaciones ocasionadas por el dataset impedirán que el modelo mejore más allá de este punto.

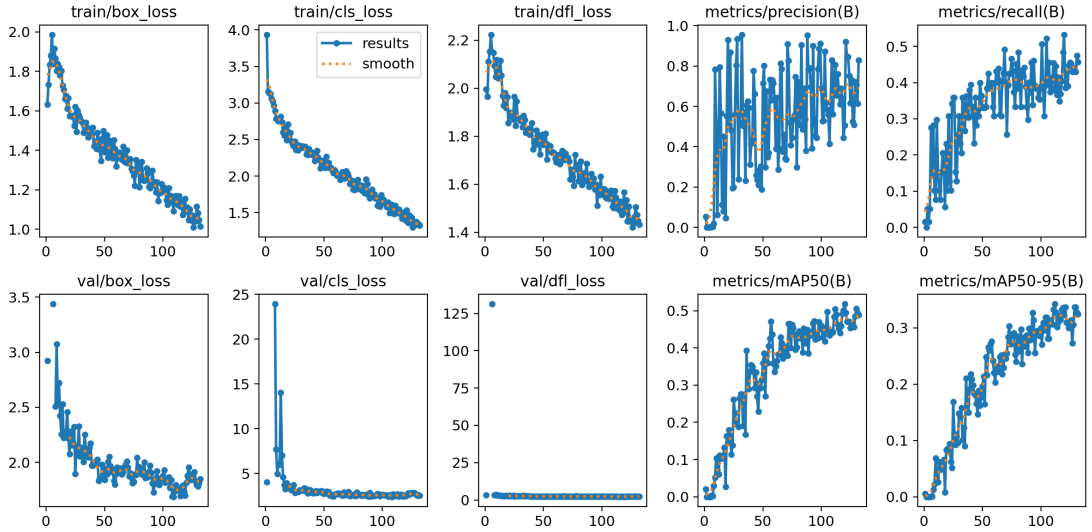

Figura 5.9. Comparación de métricas de rendimiento y de perdida para el entrenamiento (train) y para la validación (val) para 150 épocas/iteraciones. Fuente: elaboración propia.

Las gráficas de la Fig 5.12 muestran tendencia de decrecimiento para la perdida y de crecimiento para el rendimiento, con bajo nivel de fluctuación para la perdida y con nivel moderado a alto de fluctuación para las métricas de rendimiento. En este caso las gráficas de validación val/cla_loss y val/dfl_loss se mantienen bajas, pero con valores predominantemente constantes durante las 150 iteraciones, lo que resalta nuevamente que el modelo tiende al overfitting, lo cual no es positivo.

Figura 5.10. Muestra de aciertos (verdaderos positivos) y desaciertos (falsos postivos) para yolo11m para el dataset de enfermedades en la piel de perros y gatos para 150 épocas/iteraciones. Fuente: YOLO, elaboración propia

La figura 5.10 muestra como a pesar de la disminución de los valores obtenidos, la detección mejoro levemente respecto a la cantidad de detecciones obtenidas por el modelo durante el entrenamiento, lo cual es una buena señal. Sin embargo, falla en la detección de las imágenes 6 (demodicosis), 7 (fungal infection) y 8 (demodicosis), siendo estos tres casos de falsos positivos.

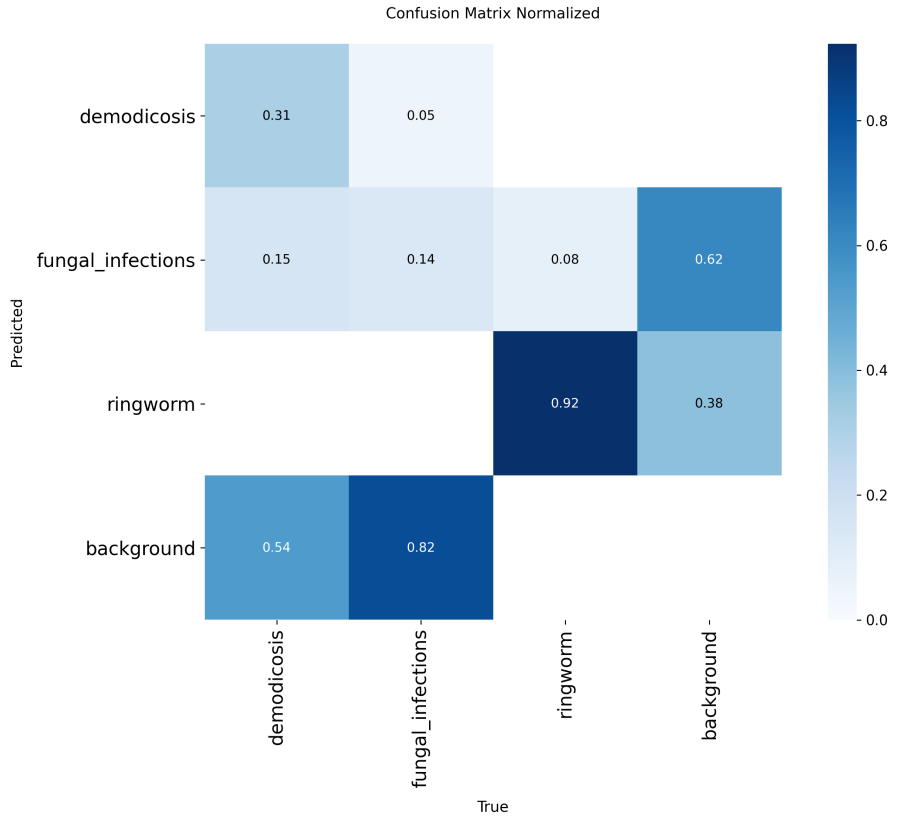

Figura 5.11. Matriz de confusión normalizada para yolo11m, mostrando las coincidencias y el porcentaje de acierto del modelo por clase, siendo background (denominación de yolo) el fondo de las imágenes del dataset. Vertical (valores predichos) vs horizontal (valores reales). Fuente: YOLO, elaboración propia.

Los resultados de la matriz de confusión normalizada indican que:

- **Demodicosis**: El acierto bajo a 31 % y el fallo aumento a 69 %.
- **Fungal infections:** el índice de acierto aumento a 14 % y los fallos disminuyeron al 86 %.
- **Ringworm**: El acierto aumento un 92 % y el fallo bajo a 8 %.

Estos valores muestran como los resultados del entrenamiento y de la validación afectan a los resultados predicciones sobre la carpeta test, por lo que a pesar del overfitting, se denota el modelo se puede mejorar resolviendo los problemas mencionados en secciones anteriores del presente capítulo.

5.8.3. Prueba con YOLOv11s (YOLO11s).

En los resultados mostrados en la tabla de la Fig 5.3, donde se puede apreciar una mejoría general el mAP50-95 con 0.378, que no dista mucho de los resultados anteriores, pero la brecha es un poco mayor en lo que respecta a esta métrica. También se puede apreciar como los valores de precisión bajaron, mientras que los de recall en general aumentaron un poco. Por otro lado, en lo que respecta mAP50, todos los valores aumentaron en general, siendo una mejora leve pero notable respecto a los otros dos modelos. Esto es gracias a la regularización

L2 y a los parámetros de aumentación añadidos.

			Box(P R mAP50 mAP50-95): 100% ———— 2/2 1.8it/s 1.1s2.8s			
Class	**Images**	**Instances**	**P**	**R**	**mAP50**	**mAP50-95**
all	40	48	0,71	0,481	0,541	0,378
demodicosis	10	13	0,798	0,462	0,527	0,327
fungal_infections	8	22	0,499	0,136	0,168	0,0797
ringworm	12	13	0,833	0,846	0,928	0,726

Tabla 5.3. Tabla de resultados de entrenamiento para el modelo yolo11s, para 150 épocas/iteraciones sobre el dataset de enfermedades de la piel en perros y gatos (Dog_skin_disease_modify). Muestra las clases (enfermedades por separado y en conjunto), la cantidad de imágenes, las instancias (cantidad de cajas) y las métricas de rendimiento: Box Precision (P), Box Recall (R), mean average precision con umbral IoU de 0.50 y de 0.50 a 0.95 (mAP50 y mAP50-95 respectivamente). Fuente: Elaboración propia.

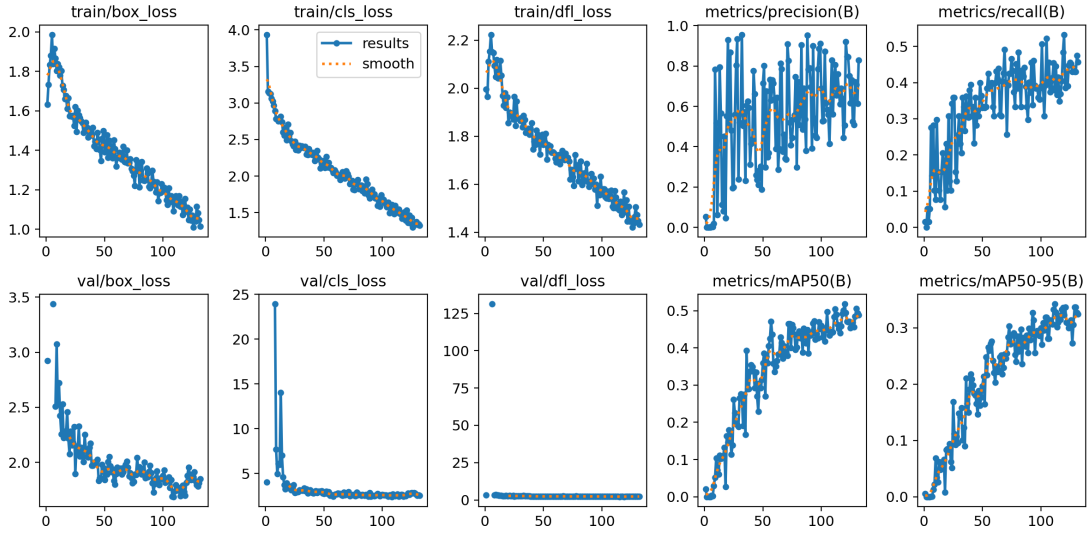

Figura 5.12. Comparación de métricas de rendimiento y de perdida para el entrenamiento (train) y para la validación (val) para 150 épocas/iteraciones para el modelo yolo11s. Fuente: elaboración propia.

Como se puede apreciar en la figura anterior, la perdida mantiene la tendencia a decrecer y rendimiento a aumentar. Sin embargo, las últimas dos gráficas de validación continúan mostrando una tendencia fuerte al sobre ajuste del modelo.

Figura 5.13. Muestra de aciertos (verdaderos positivos) y desaciertos (falsos positivos) para yolo11s para el dataset de enfermedades en la piel de perros y gatos para 150 épocas/iteraciones. Fuente: YOLO, elaboración propia.

Para el caso de los verdaderos positivos, los resultados distan mucho de los obtenidos por el modelo yolo11m. De igual manera ocurre con los resultados de falsos positivos. Sin embargo, cabe destacar que en algunos casos donde detecta la enfermedad no encierra o etiqueta todas las zonas con muestras visibles de la enfermedad como lo es el caso de la imagen 5 de la Fig 5.13.

Por otra parte, la matriz de confusion normalizada es la que aporta una mejor vista respecto al desempeño del modelo para las predicciones, como se muestra en la Fig 25, a continuación.

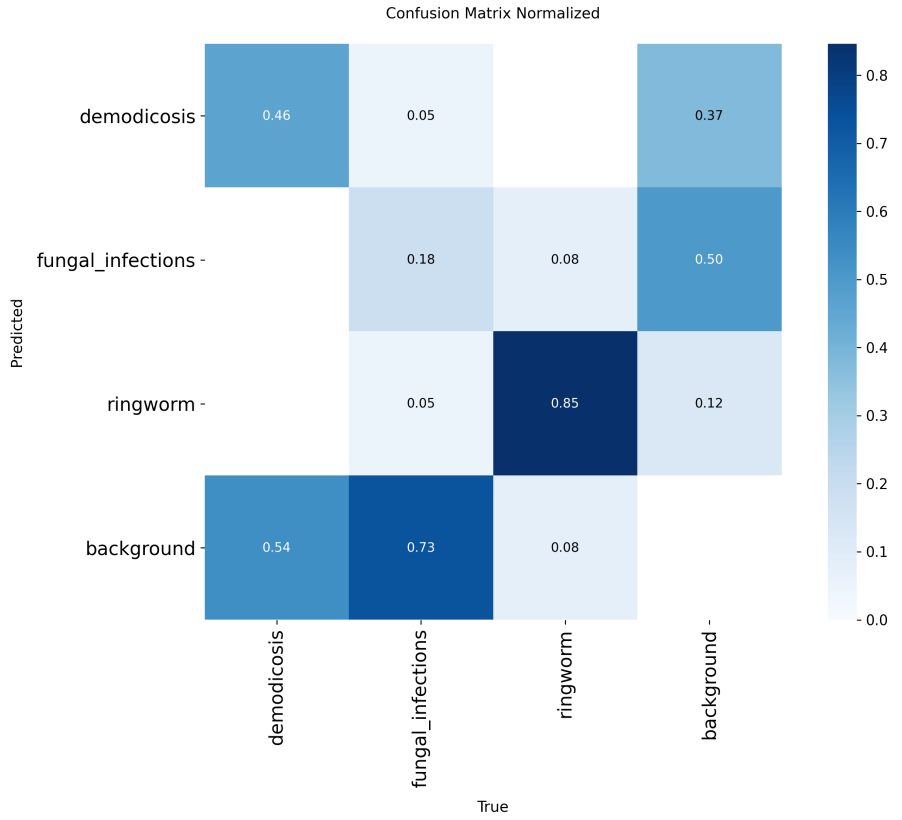

Figura 5.14. Matriz de confusión normalizada del modelo yolo11s, mostrando las coincidencias y el porcentaje de acierto del modelo por clase, siendo background (denominación de yolo) el fondo de las imágenes del dataset. Vertical (valores predichos) vs horizontal (valores reales). Fuente: YOLO, elaboración propia.

Los resultados de la matriz de confusión normalizada indican que:

- **Demodicosis**: El acierto aumento a 38 % y el fallo bajo a 62 %.
- **Fungal_infections**: el índice de acierto bajo a 5 % nuevamente y los fallos aumentaron al 95 %.
- **Ringworm**: El acierto se mantiene en 92 % y el fallo bajo a 8 %, confundiendo caracteristicas solo con el background y no con otras enfermedades.

Si bien este modelo muestra varias mejorías leves respecto a los modelos anteriores, es necesario tener en cuenta que el factor crítico principal del dataset, es decir la clase fungal_infection, vuelve a disminuir su tasa de acierto a 5 %, lo que respecto al modelo yolo11m es un decremento del 9 %.

5.8.4. Comparación de los modelos yolo11n, yolo11s y yolo11m.

Como se puede apreciar en la tabla de la Fig 26, que resume el porcentaje de aciertos para cada modelo respecto a cada enfermedad, el modelo Yolo11m es el que muestra los mejores resultados a nivel de predicción después del entrenamiento y si bien no todos los resultados son óptimos, permiten continuar con la siguiente fase prueba que combina los mejores resultados

de yolo para detección y de convNeXt para clasificación, así como el determinar la efeciencia de este proceso.

Modelo	Demodicosis	Fungal_infections	Ringworm
Yolov11n	38%	5%	85%
Yolov11s	38%	5%	92%
Yolov11m	31%	14%	92%

Tabla 5.4. Tabla comparativa entre los modelos de prueba: yolo11n (nano), yolo11s (small) y yolo11m (medium), en función de los resultados de sus respectivas matrices de confusión normalizadas. Fuente: elaboración propia.

5.8.5. Implementación de inferencia en cascada (Cascade o Cascading Inference).

Este método consiste en incluir múltiples etapas de procesamiento secuencial para refinar y mejorar la predicción, lo cual se puede lograr a través de una secuencia de detectores, donde cada uno de estos se entrena usando la salida (Bounding Boxes refinadas) del detector anterior, como propone [7]. De manera Análoga, este caso se pretende usar la salida de Yolo como entrada para el modelo ConvNeXt-tiny para lograr un clasificación refinada y contextual, usando el dataset animals ij5d2 [5] adaptado previamente para clasificación.

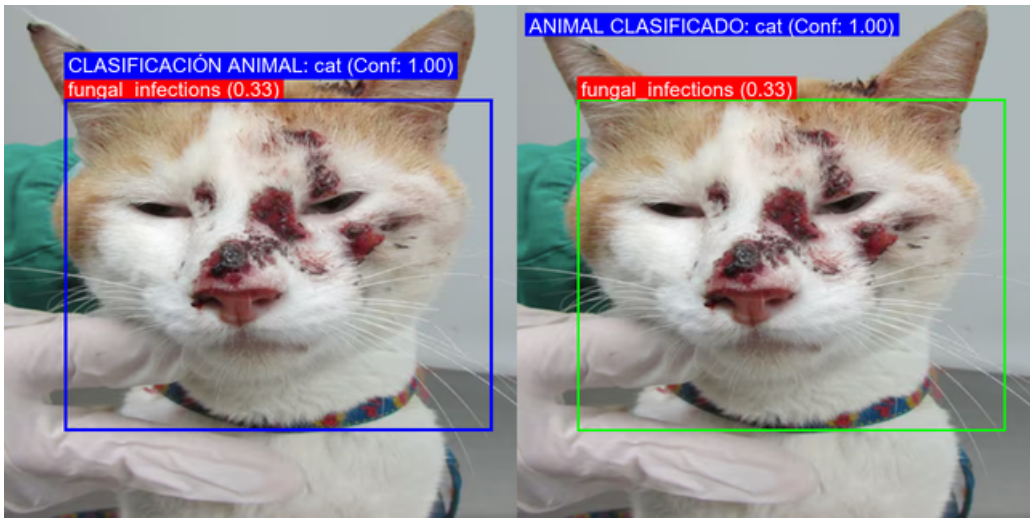

Figura 5.15. Resultado de la inferencia en cascada, con cuadro delimitador denotando el animal y el tipo de enfermedad. A la izquierda: imagen generada por el código original de inferencia. A la derecha: variación del código original para separar el cuadro general de detección de la etiqueta de clasificación. Fuente: elaboración propia.

En la Fig 5.15, se puede apreciar como a pesar de que el porcentaje de acierto para la enfermedad fungal_infections es considerablemente baja, en el caso de una imagen con buena calidad, se observa como la predicción de la zona con la enfermedad es acertada. Por otro lado, en lo que respecta a la clasificación del animal también el resultado fue preciso. Sin embargo, dado que el dataset animals ij5d2 [5] contiene 11 clases y que el modelo convNeXt-tiny posee su

propio margen de error de clasificación, se pueden obtener resultados como el que se muestra en la Fig 5.16.

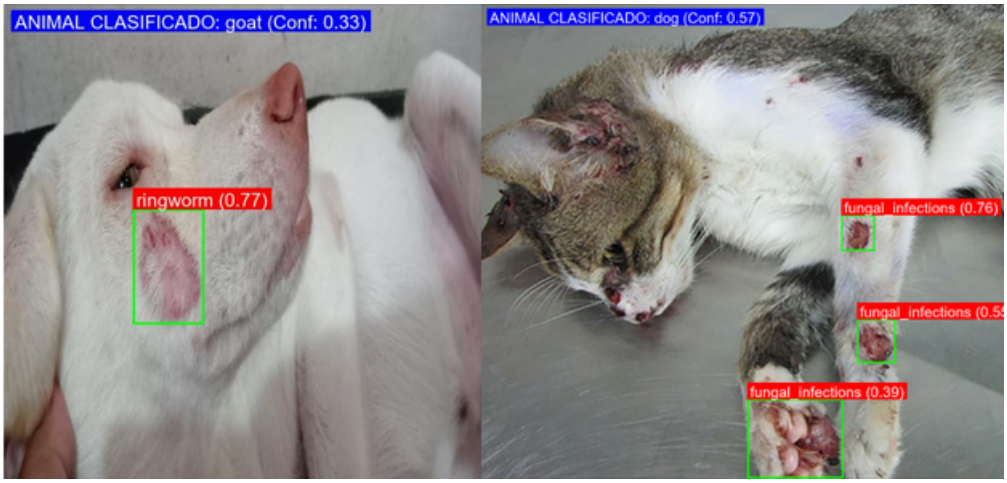

Figura 5.16. Resultados de inferencia en cascada. A la izquierda: perro clasificado como cabra (goat) con detección de enfermedad correcta. A la derecha: gato clasificado como perro (dog) con detección parcial, pero acertada de enfermedad. Fuente: elaboración propia.

Este problema se puede corregir forzando al dataset a que tome solo dos de las once etiquetas, pero implicaría agregar una capa de complejidad innecesaria para este punto, por lo que la solución más factible a largo plazo es la de utilizar un dataset orientado a clasificación que solo contenga las clases "cat" para gato y "dog" para perro, como lo es el dataset Cat_and_Dog [24] a través del cual se pueden mejorar los resultados de la clasificación a solo contener las dos clases que se adecúan a los animales que aparecen en el dataset de enfermedades.

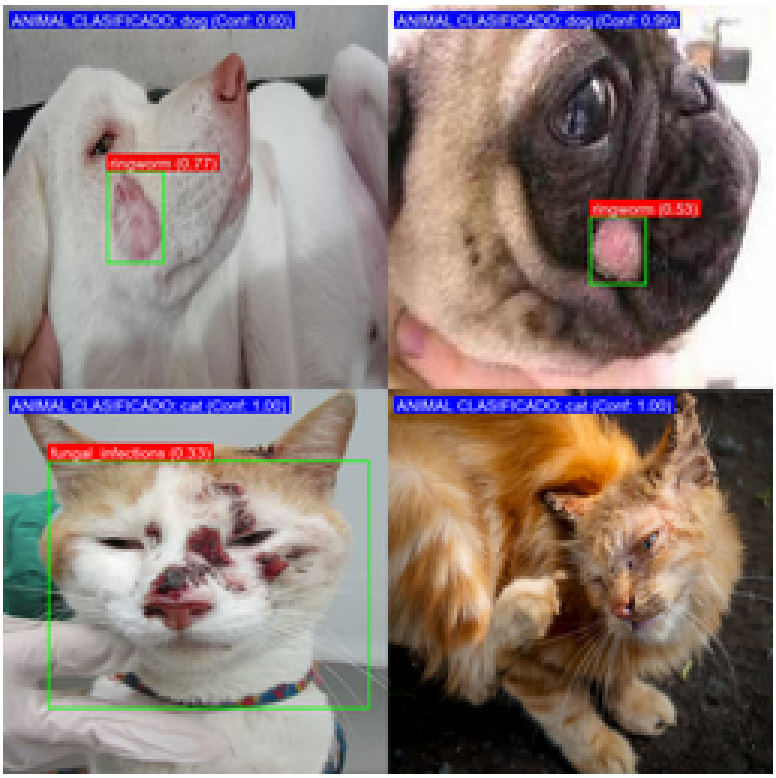

Figura 5.17. Resultados inferencia en cascada, con clasificación acertada en las cuatro imágenes, mientras que la detección de enfermedades acierta en tres de las cuatro imágenes. Fuente: elaboración propia.

5.8.6. Resumen Global del análisis de fallos.

Una vez comprobado que ambos modelos funcionan para el procesamiento de sus respectivas tareas (YOLO: Detección y ConvNeXt: Clasificación) se procede a realizar un diagnóstico general de fallo basado en la confianza (conf), ya que como se ha podido apreciar en las imágenes de resultados anteriores tanto las etiquetas de clasificación como las de las bounding boxes muestran un atributo de confianza la cual es la probabilidad predicha de que la etiqueta sea correcta para una instancia dada (una BBox, un recorte o una imagen). Es la salida de la capa softmax.

```
================================================================================
RESUMEN GLOBAL DEL ANÁLISIS DE FALLOS
================================================================================
Total de imágenes procesadas: 40
Total de casos débiles/fallidos encontrados: 43

Distribución de Casos Débiles/Fallidos (Fallos en el Pipeline):
  > FALLO_TOTAL (YOLO N/D)    | Casos: 16 | Porcentaje (aprox): 40.00%
  > ETAPA 1 DÉBIL (YOLO)      | Casos: 24 | Porcentaje (aprox): 60.00%
  > CRÍTICO (Ambos Débiles)   | Casos: 3 | Porcentaje (aprox): 7.50%
```

Figura 5.18. Resultados del "Resumen global del análisis de fallos" por cada etapa, mostrando la proporción de casos débiles, fallidos y críticos entre las 40 imágenes de validación. Fuente: elaboración propia.

Estos resultados concuerdan con las métricas obtenidas tras el entrenamiento y validación del

modelo YOLO para el dataset de enfermedades modificado el cual limita mucho el desempeño del modelo debido a la calidad del dataset. A continuación, se muestran los casos de fallo de cada modelo (YOLO y ConvNeXt).

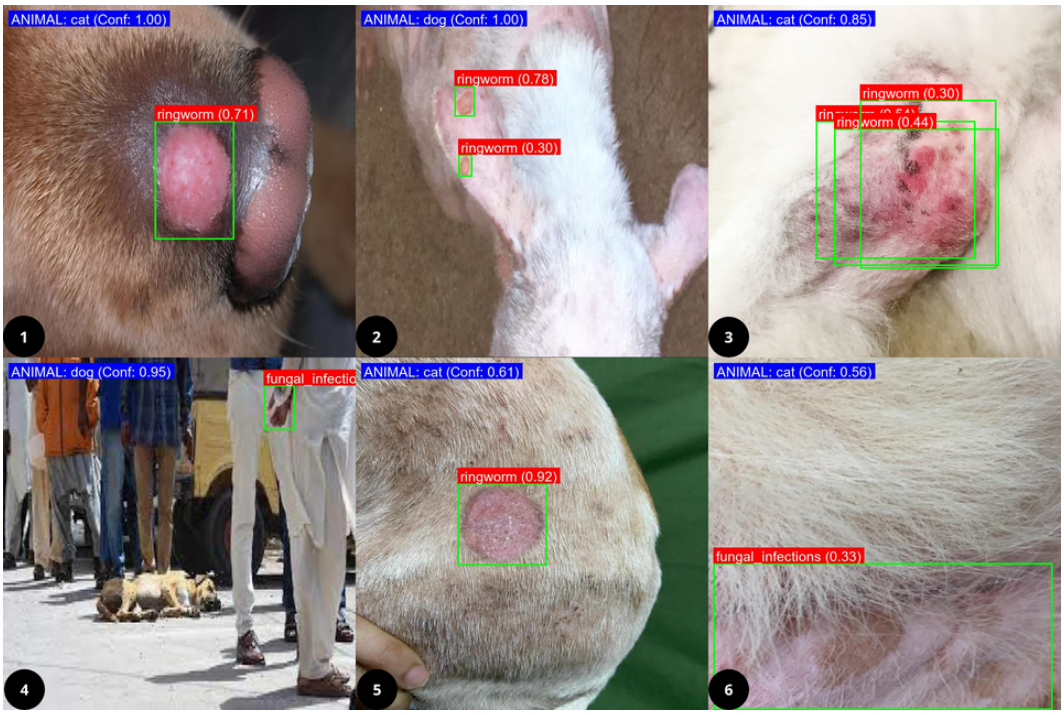

Figura 5.19. Muestra de seis casos donde al menos uno de los modelos falla en el proceso de detección de clasificación, según sea el caso. Fuente: elaboración propia

En la Fig 5.19, se pueden apreciar seis casos donde los modelos fallan al predecir el tipo de animal o la enfermedad, o ambas incluso. En lo que respecta a la clasificación se puede apreciar como en imágenes como 1, 5 y 6, donde no se puede apreciar el animal como tal, ConvNeXt-tiny presenta problemas para clasificar al animal, mostrando además una baja confianza en las etiquetas. Por otro lado, en lo que respecta a la detección de la enfermedad, hay casos como el de la imagen 4, donde detecta parte del fondo (background) como una infección fúngica, o en casos como los de las imágenes 3 y 6, donde la detección de la enfermedad es errónea en imágenes en lesiones visualmente similares. Por último, también hay casos como los de la Fig 31, donde ninguno de los dos modelos es capaz de predecir nada en sus respectivas tareas.

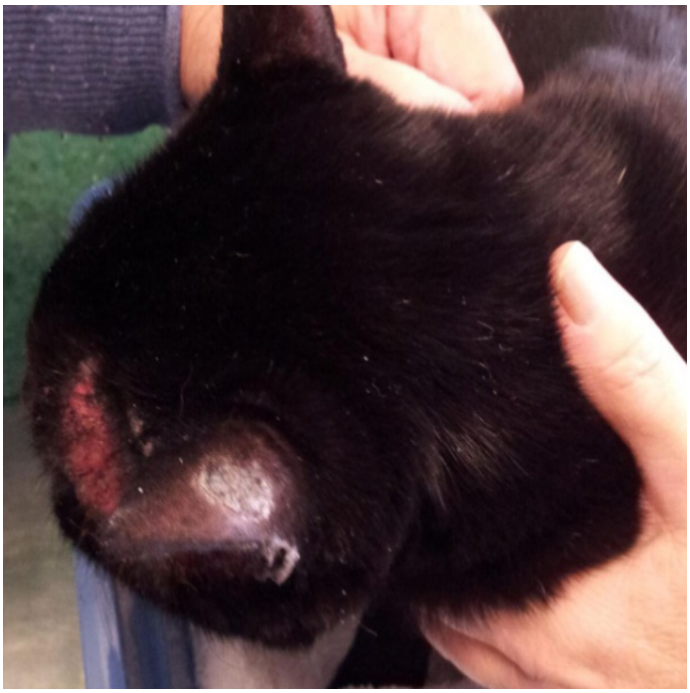

Figura 5.20. Resultado donde ambos modelos fallaron en la predicción al no detecta nada en la imagen, ni el animal "cat", ni la enfermedad "demodicosis".

5.8.7. Conclusión de las pruebas

Gran parte de los problemas de cálidad del dataset [6] provienen del hecho de que la versión disponible de este en la web Kaggle, es una versión preprocesada, donde la mayoría de las imágenes son el resultado de dicho preprocesamiento y de diversas técnicas de aumentación. De hecho, al tomar una muestra significativa, pero con menor grado de corrupción a raíz del preprocesamiento, reducir la cantidad de enfermedades listada y al aportarle variedad al incluir gatos diagnosticados con las mismas enfermedades de las clases, los resultados iniciales de detección mejoraron significativamente hasta alcanzar el límite de mejora reflejado en resultados previos.

Estos resultados, si bien no son precisamente los mejores que se podrían obtener, la tendencia decreciente de la perdida, así como la tendencia creciente en las métricas de rendimiento, reflejan que el modelo en sí mismo tiende a mejorar con el paso del tiempo, o agregándole factores de aumentación de datos que incorporen más variedad sin dañar las anotaciones realizadas, asi como la aplicación de tasas de aprendizaje dinámicas que se ajusten por época, o el early stopping, además de técnicas de ajuste, que permiten la mejora de hiperparámetros que influyan positivamente en los modelos generados.

En conclusión, el uso de un dataset de mejor calidad, con imágenes donde se pueda distinguir tanto el animal como la enfermedad, así como una cantidad de imágenes ig ual o superior a las 400 utilizadas durante el presente estudio, son factores que influiran positivamente en el desempeño de los modelos generados por YOLOv11, ya que esta versión optimiza particularmente los resultados de las métricas de rendimiento. Ahora en lo que respecta al dataset

de clasificación, lo ideal es que contenga las clases relacionadas con los tipos de animales del dataset de enfermedades, o en su defecto, estructurar el dataset de enfermedades para que contenga todos o al menos una parte de los animales que aparecen el conjunto de clasificación, pues como se pudo apreciar en las pruebas realizadas con el dataset de [5], a mayor cantidad de clases para la clasificación, aumentan las posibilidades de clasificaciones erróneas particularmente en pipelines de cascada como el usado durante estas pruebas.

Capítulo 6. DISCUSIÓN

Teniendo en cuenta los resultados anteriores, así como todas las complicaciones ocasionadas por el tipo de recursos disponibles para el procesamiento, así como las limitaciones los datasets públicos disponibles para el presente estudio. Se recalca la importancia de usar sistemas operativos como mac o Linux, dado que la gestión de recursos de windows puede entorpecer la fase de pruebas o generar errores problemáticos durante esta. Sin embargo, la tendencia a mejorar de los modelos a pesar de los problemas que se presentaron es un factor resaltante pues indica que corrigiendo los detalles antes mencionados y adicionalmente implementando servicios que permitan mejorar la capacidad de procesamiento, el modelo no solo se volverá mucho más preciso en las predicciones, sino que permitirá aumentar la cantidad de enfermedades a detectar o, inclusive desarrollar variaciones con diferentes aspectos de la detección de enfermedades que permita el generar predicciones o diagnósticos parciales más precisos que puedan ser de utilidad en campos como la veterinaria, la zoología, etc.

Ante datasets de buena calidad, pero sin anotaciones para las bounding boxes, el uso de herramientas de anotación como label-studio [21] son especialmente útiles si se dispone de conocimiento previo sobre el manejo de la herramienta, así como de la manera correcta de realizar las anotaciones para evitar introducir un factor de sobreajuste (overfitting) adicional al modelo a entrenar. En otros casos, es recomendable usar datasets preentrenados con sus respectivas anotaciones para ahorrar tiempo de trabajo y complicaciones adicionales, pues el realizar un correcto proceso de creación de anotaciones resulta en uno de los pasos más complicados y laboriosos de un proyecto de Deep Learning independientemente del formato de las anotaciones escogidas.

Entre ***las expectativas relacionadas*** al impacto del proyecto, se espera que este contribuya en el avance del uso e implementación de redes neuronales con backbones para tareas de detección de objetos, específicamente en contextos aplicados u orientados a la observación y análisis del comportamiento animal. En este contexto, el poseer la capacidad de brindar una identificación precisa y correcta de patrones anómalos, e inclusive de indicios de enfermedad en animales a partir de imágenes, abre la posibilidad de no solo expandir el input recibido a videos, sino que permitiría la implementación de soluciones automatizadas en sectores como: la veterinaria, la biología, la conservación de animales, la ganadería e incluso en el monitoreo ambiental.

De esta manera, también se abren otras posibilidades en lo que respecta a la profundización, mejora y optimización de los métodos implementados con la finalidad de expandir las aplicaciones de estos en otros ámbitos, ya que el uso de modelos pre-entrenados o backbones junto a técnicas más recientes de computer visión, como lo es el uso de redes convolucionales profundas optimizadas con Pytorch, son consideraciones que permiten acortar el tiempo de entrenamiento así como el reducir la necesidad de tener grandes volúmenes de datos anotados desde cero. Por otra parte, la implementación de técnicas como el early stopping así como la reducción dinámica del learning rate o tasa de aprendizaje, permiten aportar robustez al entrenamiento y mejorar la generalización del modelo.

Por el momento, además del uso de recursos, se prevé que uno de ***los principales desafíos*** del presente proyecto, como se mencionó anteriormente se encuentra relacionado con la calidad y la adecuación de los datasets disponibles para estudios de este tipo, así como el formato de las anotaciones (VOC, YOLO, COCO, etc.) que suelen requerir de transformaciones para poder adaptarse al framework de Pytorch, adoptando salidas tensoriales. Por otra parte, otro desafío importante a considerar es el que respecta a la interpretación de los patrones anómalos encontrados, para los cuales puede ser necesario que sean validados por criterios objetivos, desarrollados a partir de fuentes confiables, que podría requerir de una fase exploratoria adicional para definir con exactitud qué se considera realmente una anomalía, así como qué se considera una enfermedad respecto a lo que se pueda apreciar visualmente en las imágenes, particularmente en aquellas que durante las fases iniciales o durante del tratamiento de la enfermedad comparten una gran cantidad de similitudes visuales entre enfermedades.

Entre otras cosas, es importante tener en cuenta que ***una limitación relevante*** puede estar asociada al acceso a datasets suficientemente diversos y bien anotados, porque a pesar de que Hugging Face ofrece numerosas bases de datos públicas, no todas presentan la granularidad necesaria para determinar comportamientos anómalos con alta fiabilidad. Además, dado que el sistema propuesto se basa en datos visuales, podrían excluirse otras señales relevantes como lo son la temperatura, la frecuencia cardíaca, el comportamiento sonoro, entre otras características o síntomas a considerar, lo que consecuentemente restringe parcialmente el diagnóstico aportado.

Ahora en lo que respecta a las ***proyecciones futuras***, en función de los resultados, se podría considerar el ampliar el estudio hacia una línea de investigación doctoral enfocada en la detección multimodal de patologías animales, integrando señales visuales y no visuales dentro del conjunto de datos a considerar. También podría contemplarse la integración de estos sistemas en dispositivos IoT para monitoreo en tiempo real, así como el desarrollo de aplicaciones para el público común que permita tomar acciones preventivas respecto a comportamientos anómalos o extraños en sus mascotas, teniendo acceso adicional a información sobre la patología detectada, así como información o un directorio de veterinarios o especialistas en su zona residencia a los que pueda acudir para atención preventiva.

Capítulo 7. CONCLUSIONES

7.1 Conclusiones del trabajo

Este proyecto se enmarca en un campo de aplicación emergente donde confluyen la visión artificial, la inteligencia artificial aplicada y la salud animal. Se espera que los resultados obtenidos contribuyan a optimizar la detección temprana de anomalías en animales mediante imágenes, generando valor tanto en el ámbito académico como en aplicaciones prácticas.

El uso de backbones y arquitecturas profundas en un entorno bien definido como PyTorch permite abordar tareas de detección de objetos de manera eficiente. En este contexto, se han realizado pruebas preliminares con distintos modelos backbone preentrenados, destacando el buen desempeño del modelo ConvNeXt-Base en tareas de clasificación de imágenes de animales. Por el contrario, ConvNeXt-Tiny presentó limitaciones en clasificación, aunque mostró potencial para tareas de detección al ser utilizado como extractor de características mediante una arquitectura modular con cabezas específicas entrenadas por separado.

Las pruebas, realizadas sobre el dataset animals-ij5d2 [5]orientado a detección de objetos y posteriormente sobre el dataset Cat and Dog [24]orientado a clasificación de imágenes, permitieron el validar la viabilidad del flujo de trabajo propuesto, así como la necesidad de adaptar la complejidad del modelo a los recursos disponibles. La estrategia de congelamiento del backbone y entrenamiento de cabezas personalizadas se plantea como una solución eficaz para optimizar el uso computacional sin comprometer la precisión.

Sin embargo, las limitaciones a nivel de recursos y de calidad del dataset de enfermedades [6]implicaron la modificación a partir de una muestra con importe de variaciones para incorporar un segundo tipo de animal y una porción de imágenes con mejor calidad de otras fuentes públicos; también implicaron la anotación manual de las imágenes a través de label-studio y el uso del modelo YOLO para mejorar la calidad de la detección, mientras se relegaba la tarea de clasificación animal al modelo convNext-Tiny.

Esta modificación, sin descartar el uso de salidas softmax, dio paso a la implementación de la técnica de inferencia en cascada (Cascade o Cascading) en busca de aprovechar las bondades de ambos modelos para potenciar el rendimiento general de ambos. Obteniendo resultados con etiquetas de clasificación del animal y cuadros delimitadores de la enfermedad.

Para este punto, las limitaciones del dataset de enfermedades afectan considerablemente la proporción de acierto en las predicciones del modelo, particularmente en el caso de YOLO, pero mostrando a la vez una clara tendencia a mejorar, lo que implica que mejorando la calidad del dataset de enfermedades, el rendimiento general del modelo, así como su capacidad de acierto deberían aumentar significativamente puesto que el modelo funciona aun con datos de baja calidad.

A partir de estos resultados, el proyecto no solo confirma su factibilidad técnica, sino que también sienta las bases para futuras extensiones orientadas a la clasificación de enfermedades y comportamientos anómalos con mayor precisión, consolidando su aplicabilidad en contextos

reales, con la consideración de algunos de los factores de riesgo a evitar para evitar afectar el rendimiento y precisión de futuros proyectos en esta área.

No obstante, se identifican posibles desafíos técnicos en cuanto a compatibilidad de formatos de anotación, disponibilidad de datos suficientemente diversos y definición clara de lo que constituye un comportamiento anómalo.

A futuro, este trabajo podría sentar las bases para investigaciones más ambiciosas, incluyendo integración de datos multimodales y despliegue en entornos productivos o naturales.

Capítulo 8. FUTURAS LÍNEAS DE TRABAJO

Las conclusiones derivadas de este Trabajo Fin de Máster abren diversos caminos para futuras líneas de investigación que podrían consolidar y extender la aplicabilidad del sistema propuesto:

- **Mejora y Creación de Datasets de Alta Calidad:** La limitación más significativa del proyecto fue la calidad y anotación de los datos. Una línea prioritaria es la adquisición o curación de un dataset con imágenes de alta resolución y anotaciones bounding box precisas, verificadas por especialistas (veterinarios). Esto es crucial para mitigar el sobreajuste y mejorar las métricas de detección (mAP50-95) del modelo YOLOv11.
- **Optimización y Robustez del Pipeline de Detección:** Implementar estrategias de fine-tuning y ajuste de hiperparámetros más avanzados (como la búsqueda bayesiana o la optimización con frameworks automáticos) para el modelo de detección. Esto buscaría mejorar la generalización y la robustez del modelo ante la variabilidad de las imágenes. Explorar la fusión de características provenientes de backbones más recientes y específicos para visión animal, buscando modelos que equilibren mejor el poder de cómputo con la precisión.
- **Expansión del Alcance y Aplicabilidad:** Monitoreo de Comportamiento Anómalo en Video: Extender la metodología para trabajar con secuencias de video en lugar de imágenes estáticas. Esto permitiría no solo detectar enfermedades, sino también identificar comportamientos anómalos o de estrés en tiempo real, lo que es de gran valor en etología y ganadería.
- **Diagnóstico Multimodal:** Investigar la integración de datos no visuales (ej. registros médicos, datos ambientales o sonidos) con los resultados de la visión artificial para generar un diagnóstico asistido por IA más completo y preciso.
- **Despliegue e Integración en Entorno Productivo:** Desarrollar una aplicación web o móvil (frontend) que consuma el pipeline de inferencia. Esto facilitaría la integración de la herramienta como un sistema de apoyo a la decisión para veterinarios o personal de centros de rescate, permitiendo el acceso rápido y la interpretación de los resultados del modelo.

Bibliografía

[1] P. Jeevan y A. Sethi, «Which Backbone to Use: A Resource-efficient Domain Specific Comparison for Computer Vision,» mar. de 2025. dirección: http://arxiv.org/abs/2406.05612.

[2] Ultralytics., *Ultralytics YOLO: Documentación y Código*, 2024. dirección: https://docs.ultralytics.com/es/.

[3] H. Face., *Hugging Face*. dirección: https://huggingface.co/.

[4] Kaggle., *Kaggle: Your Home for Data Science*. dirección: https://www.kaggle.com/.

[5] Francesco y R. 100, *animals ij5d2 Dataset*, nov. de 2022. dirección: https://huggingface.co/datasets/Francesco/animals-ij5d2.

[6] Y. Mohmmed, *Dogs Skin Diseases Image Dataset [Data set]*, 2023. dirección: https://www.kaggle.com/datasets/youssefmohmmed/dogs-skin-diseases-image-dataset.

[7] Z. Cai y N. Vasconcelos, «Cascade R-CNN: Delving into High Quality Object Detection,» dic. de 2017. dirección: http://arxiv.org/abs/1712.00726.

[8] B. Macukow, «Neural networks-state of art, brief history, basic models and architecture,» en *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9842 LNCS, Springer Verlag, 2016, págs. 3-14, ISBN: 9783319453774. DOI: 10.1007/978-3-319-45378-1_1.

[9] R. K. Sinha, R. Pandey y R. Pattnaik, «Deep Learning For Computer Vision Tasks: A review,» inf. téc., 2017.

[10] D. W. Otter, J. R. Medina y J. K. Kalita, «A Survey of the Usages of Deep Learning in Natural Language Processing,» dic. de 2019. dirección: http://arxiv.org/abs/1807.10854.

[11] R. J. Frank, N. Davey y S. P. Hunt, «Time Series Prediction and Neural Networks,» inf. téc., 2001, págs. 91-103.

[12] M. Duguleana y G. Mogan, «Neural networks based reinforcement learning for mobile robots obstacle avoidance,» *Expert Systems with Applications*, vol. 62, págs. 104-115, nov. de 2016, ISSN: 09574174. DOI: 10.1016/j.eswa.2016.06.021.

[13] D. Z. Liu y G. Singh, «A Recurrent Neural Network Based Recommendation System,» inf. téc. dirección: https://www.yelp.com/dataset_challenge.

[14] T. K. Paradarami, N. D. Bastian y J. L. Wightman, «A hybrid recommender system using artificial neural networks,» *Expert Systems with Applications*, vol. 83, págs. 300-313, oct. de 2017, ISSN: 09574174. DOI: 10.1016/j.eswa.2017.04.046.

[15] Y. Xin, J. Yang, S. Luo et al., «Parameter-Efficient Fine-Tuning for Pre-Trained Vision Models: A Survey,» mar. de 2025. dirección: http://arxiv.org/abs/2402.02242.

[16] M. H. M. Noor y A. O. Ige, «A Survey on State-of-the-art Deep Learning Applications and Challenges,» jul. de 2025. DOI: 10.1016/j.engappai.2025.111225. dirección: http://arxiv.org/abs/2403.17561%20http://dx.doi.org/10.1016/j.engappai.2025.111225.

[17] S. Gao, I. W.-H. Tsang y L.-T. Chia, «LNCS 6314 - Kernel Sparse Representation for Image Classification and Face Recognition,» inf. téc., 2010.

[18] M. Goldblum, H. Souri, R. Ni et al., «Battle of the Backbones: A Large-Scale Comparison of Pretrained Models across Computer Vision Tasks,» nov. de 2023. dirección: http://arxiv.org/abs/2310.19909.

[19] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell y S. Xie, «A ConvNet for the 2020s,» mar. de 2022. dirección: http://arxiv.org/abs/2201.03545.

[20] MakeSense.ai., *MakeSense.AI: Free to use online data labelling tool*, 2024. dirección: https://www.makesense.ai/.

[21] Heartex., *Label Studio: An Open Source Data Labeling Tool*, 2024. dirección: https://labelstud.io/.

[22] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi y N. Smith, «Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping,» feb. de 2020. dirección: http://arxiv.org/abs/2002.06305.

[23] R. Padilla, S. L. Netto, E. A. B. D. Silva y S. L. Netto, «A Survey on Performance Metrics for Object-Detection Algorithms,» 2020. DOI: 10.1109/IWSSIP48289.2020. dirección: https://www.researchgate.net/publication/343194514.

[24] Bingsu, «Cat and Dog [Data set],» 2023. dirección: https://huggingface.co/datasets/Bingsu/Cat_and_Dog (visitado 10-11-2025).

Capítulo 9. ANEXOS

9.1 Codigo 1: Resultados para ConvNeXt-B para clasificación

Code

```
# 0. Importacion de librerias
from transformers import ConvNextImageProcessor, ConvNextForImageClassification
import torch
from datasets import load_dataset
import matplotlib.pyplot as plt

# 1. Cargar el dataset
dataset = load_dataset("Francesco/animals-ij5d2")
images = dataset["test"]["image"]

# 2. Cargar el procesador y modelo
processor =ConvNextImageProcessor.from_pretrained("facebook/convnext-base-224-22k")
model =ConvNextForImageClassification.from_pretrained("facebook/convnext-base-224-22k")
model.eval()

# 3. Seleccionar el número de imágenes
num_images = 6
cols = 3
rows = (num_images + cols - 1) // cols

# 4. Crear la figura
fig, axes = plt.subplots(rows, cols, figsize = (15, 8))
axes = axes.flatten()

# 5. Procesar imágenes individualmente
for i in range(num_images):
    image = images[i]

    # Preprocesar
    inputs = processor(image, return_tensors="pt")

    # Inferencia sin gradientes
    with torch.no_grad():
        logits = model(**inputs).logits

    predicted_label = logits.argmax(-1).item()
    label = model.config.id2label[predicted_label]
```

```
    # Mostrar la imagen con el título
    ax = axes[i]
    ax.imshow(image)
    ax.set_title(f"Predicción: {label}", fontsize=10)
    ax.axis("off")

# Ocultar ejes vacíos
for j in range(num_images, len(axes)):
    axes[j].axis("off")

plt.tight_layout()
plt.show()
```

9.2 Código 2: Resultados para ConvNeXt-tiny para clasificación

Code

```
from transformers import AutoImageProcessor, ConvNextForImageClassification

# 1. Cargar el dataset
dataset = load_dataset("Francesco/animals-ij5d2")
images = dataset["test"]["image"]

# 2. Cargar el procesador y modelo
processor = ConvNextImageProcessor.from_pretrained("facebook/convnext-tiny-224")
model = ConvNextForImageClassification.from_pretrained("facebook/convnext-tiny-224")
model.eval()  # Poner en modo evaluación

# 3. Seleccionar el número de imágenes
num_images = 6
cols = 3
rows = (num_images + cols - 1) // cols

# 4. Crear la figura
fig, axes = plt.subplots(rows, cols, figsize=(15, 8))
axes = axes.flatten()

# 5. Procesar imágenes individualmente
for i in range(num_images):
    image = images[i]

    # Preprocesar
    inputs = processor(image, return_tensors="pt")

    # Inferencia sin gradientes
    with torch.no_grad():
        logits = model(**inputs).logits

    predicted_label = logits.argmax(-1).item()
    label = model.config.id2label[predicted_label]

    # Mostrar la imagen con el título
    ax = axes[i]
    ax.imshow(image)
    ax.set_title(f"Predicción: {label}", fontsize=10)
    ax.axis("off")

# Ocultar ejes vacíos
for j in range(num_images, len(axes)):
    axes[j].axis("off")

plt.tight_layout()
plt.show()
```

9.3 Consideraciones técnicas.

Es importante tener en consideración que, si el sistema operativo del equipo usado es windows, se deben realizar ajustes en la cantidad o número de workers para evitar que el codigo falle con el siguiente error:

errorMessage

```
RuntimeError:DataLoaderworker(pid(s)18912,1264,4060,10756)exited
unexpectedly
```

El cual es un error común en Pytorch cuando la cantidad de workers definida es mayor a cero durante la carga paralela en windows, que intenta usar todos los nucleos disponibles del equipo para el procesamiento. En datasets grandes esto puede ocurrir con frecuencia, por lo que se recomienda el modificar la cantidad de workers a cero (num_workers $= 0$) en los DataLoaders para desactivar la carga paralela y de esta manera ejecutando el entrenamiento en el hilo principal, lo que trae como consecuencia un proceso de entrenamiento más lento, pero se evita elproblema antes mencionado.

Otra solución podría ser el agregar alguna condición de salto para saltar muestras del dataset no validas (sin etiquetado) y devolver una muestra validad como suele ser la práctica usual en datasets de Hugging Face, de esta manera solo se indexarían muestras válidas y se evitaría llegar al punto de lidiar con este tipo de error. También puede ocurrir que este error se haga presente por problemas de memoria o de recursos, ya que el worker se puede quedar sin memoria, o si la carga de procesamiento a la que están sometidos es intensa

9.4 Código 3: Adaptación de dataset de detección a clasificación

errorMessage

```
from datasets import load_dataset
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import ToTensor
import torch
from torchvision.transforms.functional import to_pil_image

# Cargar el dataset de Hugging Face
dataset3 = load_dataset("Francesco/animals-ij5d2")

# 1. Obtener información de clases.
id2label = dataset3["train"].features["objects"]["category"].feature.names
NUM_ANIMAL_CLASSES = len(id2label)

print(f"El dataset tiene {NUM_ANIMAL_CLASSES} clases de animales: {id2label}")
print("-" * 50)

# 2. ADAPTACIÓN DE LA CLASE DEL DATASET (SOLO CLASIFICACIÓN)
# Se asume que cada imagen tiene al menos una anotación y, sí tiene varias,
# solo se usará la primera anotación de animal para la etiqueta de la imagen.

class AnimalClassificationDataset(Dataset):
    def __init__(self, hf_dataset, transform=None):
        self.dataset = hf_dataset
        self.transform = transform
        self.to_tensor = ToTensor()

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        ejemplo = self.dataset[idx]

        # Convertir a imagen PIL para aplicar transforms fácilmente
        image_pil = ejemplo["image"].convert("RGB")

        # Se asume que el primer objeto es el animal principal a clasificar
        if not ejemplo["objects"]["category"]:
            raise ValueError(f"Muestra {idx} no tiene etiquetas de objeto.")

        # La etiqueta para la CLASIFICACIÓN es la categoría del primer objeto
        label = ejemplo["objects"]["category"][0]

        if self.transform:
            image = self.transform(image_pil)
        else:
            image = self.to_tensor(image_pil)

        # Devolver el tensor de la imagen y la etiqueta de clasificación
        return image, torch.tensor(label, dtype=torch.long)
```

9.5 Estructura de anotaciones YOLO.

consiste en un txt de la siguiente forma:

```
1 0.2839195979899497 0.2974874346300186 0.5678391959798994 0.5246231104660675
1 0.7261306532663315 0.6792294807370175 0.3098827470686764 0.35678391959798816
```

Figura 9.1. Anotaciones tensoriales en formato YOLO. Fuente: elaboración propia

donde cada linea representa el tensor de una bounding box, el primer valor se encuentra asociado al id de la clase asignada a la enfermedad, el segundo y tercer valor representan las coordenadas "x" e "y" de la caja, mientras que el cuarto y quinto valor representan el ancho "w" y el alto "h" de la caja.

YOLO Dictionary

```
{
  "categories": [
    {
      "id": 0,
      "name": "demodicosis"
    },
    {
      "id": 1,
      "name": "fungal_infections"
    },
    {
      "id": 2,
      "name": "ringworm"
    }
  ],
  "info": {
    "year": 2025,
    "version": "1.0",
    "contributor": "Label Studio"
  }
}
```

9.5.1. Documento YAML asociado

El documento datos.yaml asociado a las pruebas con YOLO es el siguiente:

data.yaml file

```
path: C:\Users\keisb\Documents\datasets
train: images\train
val: images\val
test: images\test
nc: 3
names:
  0: demodicosis
  1: fungal_infections
  2: ringworm
```

9.6 Código 4: Prueba con YOLO11n.

YOLO11n test

```
from ultralytics import YOLO

model2 = YOLO("yolo11n.pt") #CArgar modelo yolo nano, que es el mas pequeño

results2 = model2.train(
    data=r"C:\Users\keisb\Documents\tfm_code\data_yolo.yaml",
    epochs=50, # Iteraciones.
    imgsz=640, # Dimension de las imagenes (dim maxima).
    plots=True, # Guarda graficos de entrenamiento.
    weight_decay = 0.001,
    workers=0 # Evita que se active el procesamiento paralelo.
)
```

9.7 Código 5: Prueba con YOLO11m.

YOLO11m test

```
# 1. Cargamos el modelo Medium preentrenado.
model_medium = YOLO("yolo11m.pt")

# 2. Iniciamos el entrenamiento con la configuración optimizada de la Fase 1.
# La capacidad extra del modelo Medium ahora puede usar esta configuración
# para alcanzar un rendimiento superior.
results_yolo11m = model_medium.train(
    data=r"C:\Users\keisb\Documents\tfm_code\data_yolo.yaml",
    epochs=150,
    imgsz=640,
    plots=True,
    name='entrenamiento_yolo11m_primer_intento',

    # --- Parámetros de REGULARIZACIÓN Y EARLY STOPPING ---
    patience=20,
    weight_decay=0.001,   # Regularización L2

    # --- Parámetros de AUMENTACIÓN ---
    hsv_s=0.8,
    translate=0.2,
    scale=0.6,
    workers=0
)
```

9.8 Código 6: Prueba con YOLO11s.

YOLO11s test

```
# Prueba del modelo Small
model5 = YOLO("yolo11s.pt")
results_fase1 = model5.train(
    data=r"C:\Users\keisb\Documents\tfm_code\data_yolo.yaml",
    epochs=150,
    imgsz=640,
    plots=True,
    name='entrenamiento_fase1_regularizado',

    # --- Parámetros de REGULARIZACIÓN Y EARLY STOPPING ---
    weight_decay=0.001,
    patience=20,

    # --- Parámetros de AUMENTACIÓN MÁS AGRESIVA ---
    hsv_s=0.8,
    translate=0.2,
    scale=0.6,
    workers=0
)
```

9.9 Código 7: Inferencia en cascada.

Cascade Inference (Cascading)

```
import os
from ultralytics import YOLO
from PIL import Image, ImageDraw, ImageFont
import torch
import torch.nn as nn
from transformers import ConvNextV2Model
import torch.nn.functional as F
from torchvision import transforms

#=============================================================================
# 0. CONFIGURACIÓN SÍNCRONA DE CLASES Y MODELO CONVNEXT
#=============================================================================

# 1. Mapeo de Clases YOLO (Enfermedades)
YOLO_ID_TO_LABEL = {
    0: 'demodicosis',
    1: 'fungal_infections',
    2: 'ringworm'
}

# 2. Mapeo de Clases ConvNeXt (Animales - 11 clases)
CONVNEXT_ID_TO_ANIMAL = {
    0: 'animals', 1: 'cat', 2: 'chicken', 3: 'cow', 4: 'dog',
    5: 'fox', 6: 'goat', 7: 'horse', 8: 'person', 9: 'racoon',
    10: 'skunk'
}

NUM_ANIMAL_CLASSES = len(CONVNEXT_ID_TO_ANIMAL)

# Definición de la clase AnimalClassifier (BACKBONE CONVNEXT)
class AnimalClassifier(nn.Module):
    def __init__(self, backbone, num_animal_classes):
        super().__init__()
        self.backbone = backbone
        # Hidden size es 768 para ConvNextV2 Tiny
        feature_size = backbone.config.hidden_sizes[-1]
        self.classifier_head = nn.Sequential(
            nn.Linear(feature_size, 256),
            nn.ReLU(),
            nn.Linear(256, num_animal_classes)
        )
```

```
    def forward(self, x):
        features = self.backbone(x).last_hidden_state
        # FIX ROBUSTO DE POOLING: Global Average Pooling para garantizar (B, 768)
        if features.dim() == 3:
            pooled_features = features.mean(dim=1)
        elif features.dim() == 4:
            # Aplica GAP a las dimensiones H y W, dejando (B, C)
            pooled_features = F.adaptive_avg_pool2d(features, (1, 1)).flatten(1)
        else:
            raise RuntimeError(f"Unexpected feature dimension: {features.dim()}")
        logits = self.classifier_head(pooled_features)
        return logits

# ==============================================================================
# 1. CONFIGURACIÓN DE RUTAS Y CARGA DE MODELOS
# ==============================================================================

RUTA_YOLO_ENFERMEDAD = r"C:\entrenamiento_yolo11m_primer_intento\weights\best.pt"
RUTA_CONVNEXT_ANIMAL = r"C:\tfm_code\convnext_animal_classifier(tiny)_best.pth"
RUTA_IMAGEN_PRUEBA = r"C:\datasets\images\val\fcd9c6c3-5f.png"
RUTA_IMAGEN_SALIDA = r"C:\tfm_code\image_results\imagen_con_resultados.png"

# Carga YOLO
try:
    yolo_model = YOLO(RUTA_YOLO_ENFERMEDAD)
except Exception as e:
    print(f"ERROR: No se pudo cargar el modelo YOLO. Verifique la ruta: {e}")
    exit()

# Carga ConvNeXt
try:
    animal_backbone=ConvNextV2Model.from_pretrained("facebook/convnextv2-tiny-1k-224")
    convnext_animal_classifier=AnimalClassifier(animal_backbone, NUM_ANIMAL_CLASSES)
    convnext_animal_classifier.load_state_dict(torch.load(RUTA_CONVNEXT_ANIMAL))
except Exception as e:
    print(f"ERROR: No se pudo cargar el modelo ConvNeXt. Verifique la ruta: {e}")
    exit()

# Preparación
convnext_animal_classifier.eval()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
convnext_animal_classifier.to(device)
```

```
# Transformaciones para ConvNeXt (debe coincidir con el entrenamiento)
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# ==============================================================================
# 2. PROCESO DE INFERENCIA EN CASCADA
# ==============================================================================


def run_cascading_inference(image_path):
    """
    Ejecuta la detección (YOLO) seguida de la clasificación (ConvNeXt) en el BBox.
    """
    try:
        img_pil = Image.open(image_path).convert("RGB")
    except Exception as e:
        print(f"ERROR: No se pudo cargar la imagen de prueba: {e}")
        return []

    final_detections = []

    # 1. INFERENCIA YOLO
    # yolo_model(..., imgsz=640) para mantener la consistencia con el entrenamiento
    yolo_results = yolo_model(img_pil, imgsz=640, verbose=False)

    for result in yolo_results:
        if result.boxes:
            for box in result.boxes:
                # BBox y Clasificación de Enfermedad (YOLO)
                x1, y1, x2, y2 = [int(val) for val in box.xyxy[0].tolist()]
                cls_id = int(box.cls[0].item())

                confidence = float(box.conf[0].item())

                # 2. PROCESAMIENTO CONVNEXT
                # Recortamos usando el BBox de la enfermedad
                cropped_img_pil = img_pil.crop((x1, y1, x2, y2))
                input_tensor = transform(cropped_img_pil).unsqueeze(0).to(device)

                with torch.no_grad():
                    output = convnext_animal_classifier(input_tensor)
                    probabilities = F.softmax(output, dim=1)
                    animal_conf, animal_idx = torch.max(probabilities, 1)
                    animal_label = CONVNEXT_ID_TO_ANIMAL.get(animal_idx.item(), "...")
                    disease_label = YOLO_ID_TO_LABEL.get(cls_id, "...")
```

```
                    # 3. COMPILACIÓN DE RESULTADOS
                    final_detections.append({
                        "bbox": (x1, y1, x2, y2),
                        "enfermedad": disease_label,
                        "animal": animal_label,
                        "confianza_yolo": confidence,
                        "confianza_convnext": animal_conf.item()
                    })

    return final_detections

# ============================================================================
# 3. EJECUTAR INFERENCIA Y VISUALIZACIÓN
# ============================================================================
# 3.1 Ejecutar el proceso
detecciones = run_cascading_inference(RUTA_IMAGEN_PRUEBA)

if not detecciones:
    print("No se detectaron objetos. Saliendo.")
    exit()

# --- Configuración de Estilo para Visualización ---
COLOR_BBOX_ENFERMEDAD = "lime"
GROSOR_LINEA_ENFERMEDAD = 2
COLOR_TEXTO = "white"
GROSOR_LINEA_ANIMAL = 4
TAMAÑO_FUENTE = 25
COLOR_FONDO_ANIMAL = "blue"
COLOR_FONDO_ENFERMEDAD = "red"

# 3.2 Cargar fuente
try:
    # Intenta cargar Arial desde la ruta común de Windows
    font_path = os.path.join(os.environ.get("WINDIR", ""), "Fonts", "arial.ttf")
    font = ImageFont.truetype(font_path, TAMAÑO_FUENTE)
except (IOError, KeyError):
    # Usa la fuente por defecto si falla
    print("ADVERTENCIA: Usando la fuente por defecto (arial.ttf no encontrada).")
    font = ImageFont.load_default()

# 3.3 Cargar y preparar la imagen
img_draw = Image.open(RUTA_IMAGEN_PRUEBA).convert("RGB")
draw = ImageDraw.Draw(img_draw)

# 3.4 Dibujar las detecciones
max_conf_animal = -1.0
best_animal_label = ""
best_animal_bbox = (0, 0, 0, 0)
first_det_bbox = None # Usado para el BBox principal si hay múltiples detecciones
print("--- RESULTADOS FINALES DE DETECCIÓN ---")
```

```
for det in detecciones:
    x1, y1, x2, y2 = det['bbox']
    animal = det['animal']
    enfermedad = det['enfermedad']
    conf_yolo = det['confianza_yolo']
    conf_convnext = det['confianza_convnext']

    print(f"""
    BBox: ({x1}, {y1}, {x2}, {y2})
    Animal: {animal} ({conf_convnext:.2f})
    Enfermedad: {enfermedad} ({conf_yolo:.2f})
    """)

    if first_det_bbox is None:
        first_det_bbox = (x1,y1,x2,y2)

    # ----------------------------------------------------
    # A. Dibujar el BBox SECUNDARIO (Enfermedad)
    # ----------------------------------------------------

    # Dibujar la caja de la enfermedad (Verde)
    draw.rectangle(
        [(x1, y1), (x2, y2)],
        outline=COLOR_BBOX_ENFERMEDAD,
        width=GROSOR_LINEA_ENFERMEDAD
    )

    # Etiqueta de la enfermedad
    disease_label_short = f"{enfermedad} ({conf_yolo:.2f})"

    # Cálculo de tamaño
    text_bbox_d = draw.textbbox((x1, y1), disease_label_short, font=font)
    text_w_d = text_bbox_d[2] - text_bbox_d[0]
    text_h_d = text_bbox_d[3] - text_bbox_d[1]

    # Posición de la etiqueta corta (justo encima de la caja de la enfermedad)
    label_x_d = x1
    label_y_d = y1 - text_h_d - 5
    if label_y_d < 0: label_y_d = y2 + 5 # Ajuste si se sale

    # Dibujar fondo y texto para la enfermedad (Fondo Rojo)
    draw.rectangle(
        [(label_x_d, label_y_d), (label_x_d+text_w_d+10, label_y_d+text_h_d+5)],
        fill=COLOR_FONDO_ENFERMEDAD
    )
    draw.text(
        (label_x_d + 5, label_y_d + 2),
        disease_label_short,
        fill=COLOR_TEXTO,
        font=font
    )
```

```
    # ----------------------------------------------------
    # B. Identificar la mejor clasificación de Animal
    # ----------------------------------------------------
    if conf_convnext > max_conf_animal:
        max_conf_animal = conf_convnext
        best_animal_label = animal

# 3.5 Dibujar la Clasificación Principal del Animal (Usando el mejor resultado)
if best_animal_label and first_det_bbox:
    x1, y1, x2, y2 = first_det_bbox
    label_full = f"CLASIFICACIÓN: {best_animal_label}(Conf: {max_conf_animal:.2f})"

    # Cálculo de tamaño para la etiqueta completa
    text_bbox_full = draw.textbbox((x1, y1), label_full, font=font)
    text_w_full = text_bbox_full[2] - text_bbox_full[0]
    text_h_full = text_bbox_full[3] - text_bbox_full[1]

    # Posición de la etiqueta completa (Arriba del BBox principal)
    label_x_full = x1
    label_y_full = y1 - text_h_full - 30
    if label_y_full < 0: label_y_full = y2 + 50 # Ajuste si se sale

    # Dibujar la caja GRANDE (Azul) que enmarca la detección del animal
    draw.rectangle(
        [(x1, y1), (x2, y2)],
        outline=COLOR_FONDO_ANIMAL,
        width=GROSOR_LINEA_ANIMAL
    )

    # Dibujar fondo y texto para la clasificación del animal (Fondo Azul)
    draw.rectangle(
        [(label_x_full, label_y_full),
        (label_x_full + text_w_full + 10, label_y_full + text_h_full + 5)],
        fill=COLOR_FONDO_ANIMAL
    )
    draw.text(
        (label_x_full + 5, label_y_full + 2),
        label_full,
        fill=COLOR_TEXTO,
        font=font
    )

# 4. Guardar la imagen
output_dir = os.path.dirname(RUTA_IMAGEN_SALIDA)

if not os.path.exists(output_dir):
    os.makedirs(output_dir)
    print(f"\nDirectorio creado: {output_dir}")
```

```
try:
    img_draw.save(RUTA_IMAGEN_SALIDA)
    print(f"\n Imagen con detecciones guardada en: {RUTA_IMAGEN_SALIDA}")
except Exception as e:
    print(f"""
    ERROR: No se pudo guardar la imagen.
    Verifique los permisos de la carpeta de salida: {e}
    """)
```

9.10 Código 8: Resumen Global de fallos

Code

```
# 3. EJECUTAR INFERENCIA Y VISUALIZACIÓN + DIAGNÓSTICO EN BATCH

# Crear el directorio de salida si no existe
if not os.path.exists(OUTPUT_BASE_DIR):
    os.makedirs(OUTPUT_BASE_DIR)

# Filtramos solo por archivos que terminan en extensiones comunes
archivos_a_procesar = [f for f in os.listdir(IMAGENES_A_PROCESAR)
    if f.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp'))]

print(f"\n=======================================================")
print(f"""
PROCESANDO {len(archivos_a_procesar)} DIRECTORIO: {IMAGENES_A_PROCESAR}
""")
print(f"=======================================================")

# Definición de Umbrales de Confianza para el Análisis de Fallos.
UMBRAL_CONFIANZA_YOLO = 0.75
UMBRAL_CONFIANZA_CONVNEXT = 0.60
resultados_analisis = []

# Configuración de Estilo para Visualización
COLOR_BBOX_ENFERMEDAD = "lime"
GROSOR_LINEA_ENFERMEDAD = 3
COLOR_TEXTO = "white"
TAMAÑO_FUENTE = 25
COLOR_FONDO_ANIMAL = "blue"
COLOR_FONDO_ENFERMEDAD = "red"

# Cargar fuente (una vez)
try:
    font_path = os.path.join(os.environ.get("WINDIR", ""), "Fonts", "arial.ttf")
    font = ImageFont.truetype(font_path, TAMAÑO_FUENTE)
except (IOError, KeyError):
    font = ImageFont.load_default()

for image_filename in archivos_a_procesar:
    RUTA_IMAGEN_PRUEBA = os.path.join(IMAGENES_A_PROCESAR, image_filename)

    # Crear un nombre de archivo de salida único
    base_name, ext = os.path.splitext(image_filename)
    RUTA_IMAGEN_SALIDA = os.path.join(OUTPUT_BASE_DIR, f"{base_name}_resultado{ext}")

    # Ejecutar el proceso de inferencia en cascada
    detec_enfermedad, clasif_animal = run_cascading_inference(RUTA_IMAGEN_PRUEBA)
```

```
# ----------------------------------------------------------------------
# A. DIAGNÓSTICO Y ALMACENAMIENTO DE ERRORES
# ----------------------------------------------------------------------

if not detecciones_enfermedad:
    # FALLO_TOTAL
    resultados_analisis.append({
        "imagen": image_filename,
        "tipo_fallo": "FALLO_TOTAL (YOLO N/D)",
        "diagnostico": "YOLO (Etapa 1) no detectó ninguna enfermedad.",
        "confianza_yolo": 0.0,
        "confianza_convnext": 0.0
    })
else:
    # Iterar sobre las detecciones para el análisis y la visualización
    for det_idx, det in enumerate(detecciones_enfermedad):

    if det_idx < len(clasif_animal):
        animal_data = clasif_animal[det_idx]
    else:
        animal_data = {"animal": "N/A", "conf_convnext": 0.0}

        conf_yolo = det['confianza_yolo']
        conf_convnext = animal_data['confianza_convnext']
        fallo_yolo = conf_yolo < UMBRAL_CONFIANZA_YOLO
        fallo_convnext = conf_convnext < UMBRAL_CONFIANZA_CONVNEXT

        tipo_fallo = "PREDICCIÓN FUERTE"
        diagnostico = ""

        # Lógica del diagnóstico
        if fallo_yolo and fallo_convnext:
            tipo_fallo = "CRÍTICO (Ambos Débiles)"
            diagnostico = """
            La detección de enfermedad (YOLO) y la clasificación de animal
            (ConvNeXt) son débiles.
            """

        elif fallo_yolo:
            tipo_fallo = "ETAPA 1 DÉBIL (YOLO)"
            diagnostico = f"""
            La detección de enfermedad '{det['enfermedad']}' es débil.
            """
        elif fallo_convnext:
            tipo_fallo = "ETAPA 2 DÉBIL (ConvNeXt)"
            diagnostico = f"""
            La detección de enfermedad (YOLO) es fuerte,
            pero la clasificación de animal '{animal_data['animal']}'
            es débil."""
```

```
            if tipo_fallo != "PREDICCIÓN FUERTE":
                resultados_analisis.append({
                    "imagen": image_filename,
                    "bbox": det['bbox'],
                    "enfermedad": det['enfermedad'],
                    "animal": animal_data['animal'],
                    "tipo_fallo": tipo_fallo,
                    "diagnostico": diagnostico,
                    "confianza_yolo": conf_yolo,
                    "confianza_convnext": conf_convnext
                })

# Resto del coódigo de la inferencia ...
# ============================================================================
# 4. RESUMEN FINAL DEL DIAGNÓSTICO (Separado para mostrar la estadística)
# ============================================================================

print("\n\n" + "="*80)
print("RESUMEN GLOBAL DEL ANÁLISIS DE FALLOS DUROS")
print("="*80)
total_analizados = len(archivos_a_procesar)


if not resultados_analisis:
    print("Todas las predicciones obtuvieron alta confianza en ambas etapas.")
else:
    conteo_fallos = {}
    for res in resultados_analisis:
        tipo = res['tipo_fallo']
        conteo_fallos[tipo] = conteo_fallos.get(tipo, 0) + 1

    print(f"Total de imágenes procesadas: {total_analizados}")
    print(f"Total de casos débiles/fallidos encontrados: {len(resultados_analisis)}\n")
    print("Distribución de Casos Débiles/Fallidos (Fallos en el Pipeline):")

    for tipo, count in conteo_fallos.items():
        porcentaje = (count / total_analizados) * 100
        print(f"""
        > {tipo:<25} | Casos: {count} | Porcentaje (aprox): {porcentaje:.2f}%
        """)


    print("\nRECOMENDACIÓN DE ACCIÓN:")

    print("""
    Filtre los archivos de imagen correspondientes a los casos 'CRÍTICO'
    y 'ETAPA 2 DÉBIL' para la inspección visual prioritaria.
    """)
```