Detección de Ganado con Técnicas Avanzadas de Aprendizaje Profundo

Arturo Cristián Díaz López
Instituto Tecnológico y de Estudios Superiores de Monterrey

01-Jun-2025

Resumen

En este proyecto se desarrolló un sistema basado en redes neuronales convolucionales para detectar vacas en imágenes tomadas en un pasillo de establo. Se utilizaron y compararon los modelos YOLOv5 y YOLOv11, entrenados sobre un conjunto de datos etiquetado manualmente. Se documentan los métodos, experimentos y resultados obtenidos, los cuales demuestran sentar una buena base hacia la continuación y mejora es este proyecto.

1 Introduction

La ganadería es una actividad económica fundamental en muchos países, especialmente en zonas rurales donde representa una fuente importante de ingresos y alimento. En este contexto, el monitoreo del ganado es esencial para garantizar su bienestar, salud y productividad. Sin embargo, los métodos tradicionales de conteo y vigilancia suelen ser manuales, demandando tiempo y recursos humanos significativos.

Gracias a los avances recientes en visión por computadora y aprendizaje profundo, hoy es posible automatizar muchas de estas tareas mediante algoritmos de detección de objetos. En particular, la familia de modelos YOLO (*You Only Look Once*) ha demostrado ser altamente eficiente para detectar instancias de objetos en imágenes en tiempo real.

Este proyecto se centra en el desarrollo y evaluación de un sistema basado en redes neuronales convolucionales para detectar vacas en imágenes capturadas en un pasillo de establo. Para ello, se utilizaron dos versiones del modelo YOLO: YOLOv5, una versión estable y ampliamente adoptada en la comunidad, y YOLOv11, una versión experimental más reciente.

El objetivo principal del proyecto es comparar el desempeño de ambos modelos en un escenario realista, con el fin de determinar cuál ofrece mejores resultados en términos de precisión, velocidad y robustez. Este documento describe el proceso completo de desarrollo del sistema, desde la recopilación de datos hasta la evaluación de resultados, y propone líneas de trabajo futuras para continuar mejorando la detección automática de ganado.

2 Dataset

La calidad y estructura de un dataset son elementos esenciales para el entrenamiento efectivo de un modelo de aprendizaje profundo. En proyectos de visión por computadora, contar con un conjunto de datos bien etiquetado y representativo garantiza que el modelo aprenda a identificar patrones y características relevantes en las imágenes. En este proyecto, el objetivo es detectar la presencia de ganado lechero; para ello, se utilizó un dataset proporcionado específicamente para el espacio donde se tiene planeado montar el modelo: un pasillo angosto y techado utilizado para el tránsito controlado de vacas.

El conjunto de datos se compone de imágenes capturadas mediante una cámara fija instalada en un punto elevado, orientada perpendicularmente al flujo del ganado. Esta configuración asegura una vista consistente y centrada del pasillo, lo cual facilita la detección de vacas. Las imágenes fueron recolectadas en diferentes momentos del día, bajo variaciones leves de iluminación y condiciones ambientales, permitiendo una mayor robustez del modelo ante escenarios reales.

Cada imagen del dataset fue anotada manualmente utilizando el formato estándar de YOLO, el cual incluye para cada vaca detectada: una clase (en este caso, únicamente la clase “vaca”), las coordenadas normalizadas del centro del *bounding box*, y su ancho y alto relativo a la imagen.

2.1 Fuente

El dataset utilizado en este proyecto fue proporcionado directamente por el cliente *Rancho CAETEC*, quien opera una instalación ganadera dedicada a la produc-

ción de leche. Las imágenes fueron capturadas en un pasillo específico dentro de dicha instalación, con el propósito de desarrollar un sistema de monitoreo automatizado del ganado.

Esta colaboración permitió acceder a datos reales, alineados con las condiciones operativas del entorno donde se planea que el modelo sea desplegado. Al tratarse de un entorno cerrado y controlado, las imágenes poseen características comunes en cuanto a perspectiva, iluminación y distribución del espacio, lo cual representa una ventaja significativa para entrenar modelos especializados.

Cabe mencionar que el uso de datos reales y propios, en lugar de conjuntos públicos genéricos, permite desarrollar un modelo más preciso, con menor riesgo de sesgo y mejor adaptado a los desafíos prácticos del entorno donde será implementado.

Figura 1: *Ejemplo de imagen proveniente del set de datos.*

2.2 Estructura

El dataset utilizado para el desarrollo del modelo contiene lo siguiente:

- Un total de 4200 imágenes con resolución de 1920 píxeles, representando una amplia variedad de situaciones dentro del pasillo, incluyendo diferentes cantidades de ganado, vacas solas, agrupadas, en movimiento y estáticas.
- 4200 archivos de texto que contiene las anotaciones, cada uno describiendo las coordenadas precisas de las cajas delimitadoras que indican la presencia de ganado en las imágenes.

Estas anotaciones proporcionan información esencial para el modelo, permitiéndole aprender a detectar y localizar el ganado en diversas condiciones. La anotación adecuada de cada imagen asegura que el modelo pueda diferenciar entre el espacio del rancho y la presencia de vacas.

Este dataset fue dividido en conjuntos de entrenamiento, validación y prueba, con un balance entre las clases para evitar sesgos en la predicción (80 %, 10 % y 10 %,). Las imágenes se preprocesaron y redimensionaron a 640x320 píxeles para estandarizar la entrada del modelo y optimizar su rendimiento.

Figura 2: *Imagen con anotaciones usada en el set de entrenamiento.*

3 Modelo

Los modelos desarrollados para este proyecto se basa en técnicas avanzadas de aprendizaje profundo, específicamente en el ámbito de la detección de objetos. La detección de objetos permite identificar y localizar instancias de interés dentro de una imagen.

3.1 Arquitectura

YOLO

YOLO, que significa "You Only Look Once", es una arquitectura revolucionaria en la detección de objetos que ha transformado el campo de la visión por computadora. Su diseño se basa en la idea de realizar la detección en una única pasada a través de la red, lo que contrasta con métodos más tradicionales que operan en múltiples etapas. Esta característica permite que YOLO produzca resultados de detección de objetos en tiempo real.

La arquitectura original de YOLO introduce un enfoque que divide la imagen en una cuadrícula, donde cada celda de la cuadrícula es responsable de predecir las bounding boxes y las clases de los objetos que caen dentro de esa celda. Esto no solo optimiza el proceso de detección, sino que también permite que el modelo considere el contexto global de la imagen en lugar de analizar pequeñas regiones de manera aislada. Esta capacidad de ver la imagen en su totalidad le permite a YOLO realizar detecciones más coherentes y precisas.

¿Cómo funciona?

La arquitectura de YOLO se basa en la idea de realizar detección de objetos en una sola pasada a través de la red neuronal. YOLO utiliza una estructura de red neuronal que combina varias capas convolucionales para extraer características relevantes de las imágenes de entrada. La red se compone de tres etapas principales: la extracción de características, la detección de objetos y la post-procesamiento de las predicciones.

1. **Backbone** (Extracción de características): La primera etapa de YOLOv5 implica la utilización de una red de extracción de características, que se basa en un diseño eficiente que permite reducir el tamaño de la imagen a medida que se extraen características más complejas.
2. **Neck** (Detección de objetos): Después de la extracción de características, YOLOv5 divide la imagen en una cuadrícula de celdas, donde cada celda es responsable de detectar objetos en su área correspondiente. Para cada celda, el modelo predice múltiples bounding boxes y las probabilidades asociadas a cada clase. Esto incluye la posición y el tamaño de la bounding box, así como una puntuación de confianza que indica la probabilidad de que un objeto esté presente en esa celda.
3. **Head** (Predicciones y Post-Procesamiento): En esta parte, se realizan las predicciones finales. Aquí, se generan los bounding boxes y las probabilidades de clase para los objetos detectados.

Versiones

Las versiones YOLOv5 y YOLOv11 representan dos generaciones distintas dentro de la familia de modelos YOLO, ambas utilizadas y evaluadas en este proyecto.

YOLOv5, desarrollado por Ultralytics, es una implementación estable y ampliamente adoptada escrita en *PyTorch*, lo que facilita su entrenamiento, ajuste e integración. Su arquitectura modular se divide en tres componentes principales: *Backbone* (para extracción de características), *Neck* (para fusión de características, utilizando PANet) y *Head* (para predicciones). YOLOv5 se destaca por su eficiencia y versatilidad, siendo ideal para entornos con recursos limitados, como dispositivos embebidos o sistemas en tiempo real. Se probaron variantes como `yolov5s` (versión ligera) y `yolov5l` (versión más robusta), con ajustes personalizados de hiperparámetros y técnicas de *fine-tuning*.

YOLOv11, en cambio, es una versión experimental más reciente, orientada a lograr mayor precisión mediante innovaciones arquitectónicas. Introduce mejoras como mecanismos de atención, mejor manejo de escalas múltiples y estrategias avanzadas de regularización. Aunque requiere mayor capacidad de cómputo durante el entrenamiento, YOLOv11 mostró ventajas en escenarios complejos, como imágenes con oclusión parcial, iluminación variable y objetos pequeños. Su diseño modular y su enfoque en la generalización lo convierten en una alternativa prometedora para tareas de detección más exigentes.

Ambas versiones fueron entrenadas sobre el mismo conjunto de datos y evaluadas con métricas estándar, lo cual permitió realizar una comparación directa en cuanto a desempeño y aplicabilidad en el entorno objetivo del proyecto.

4 Entrenamiento

El proceso de entrenamiento de los modelos se llevó a cabo utilizando dos implementaciones distintas de la arquitectura YOLO: YOLOv5 y YOLOv11, ambas proporcionadas por Ultralytics.

Para YOLOv5 se utilizó el repositorio oficial (`https://github.com/ultralytics/yolov5`), el cual ofrece una implementación optimizada en PyTorch. Se configuró el entorno con todas las dependencias necesarias, incluyendo PyTorch, OpenCV y otras librerías auxiliares, y se habilitó la aceleración por GPU a través de CUDA.

El conjunto de datos se organizó en el formato requerido por YOLO: carpetas separadas para entrenamiento, validación y prueba, junto con un archivo `data.yaml` que define las rutas y clases.

El comando utilizado para iniciar el entrenamiento fue el siguiente:

```
python train.py --img 640 --batch 4
--epochs 50 --data ../data.yaml
--weights yolov5s.pt --device 0
```

En este comando, se utilizó una resolución de imagen de 640 píxeles, un tamaño de lote de 4, y un total de 50 épocas. Se partió de los pesos preentrenados `yolov5s.pt`, lo que permitió una convergencia más rápida y eficaz.

Durante el entrenamiento, se monitorearon métricas clave como la pérdida de clasificación, localización y confianza, así como la *mean Average Precision* (mAP) en el conjunto de validación. Al finalizar, se guardaron los

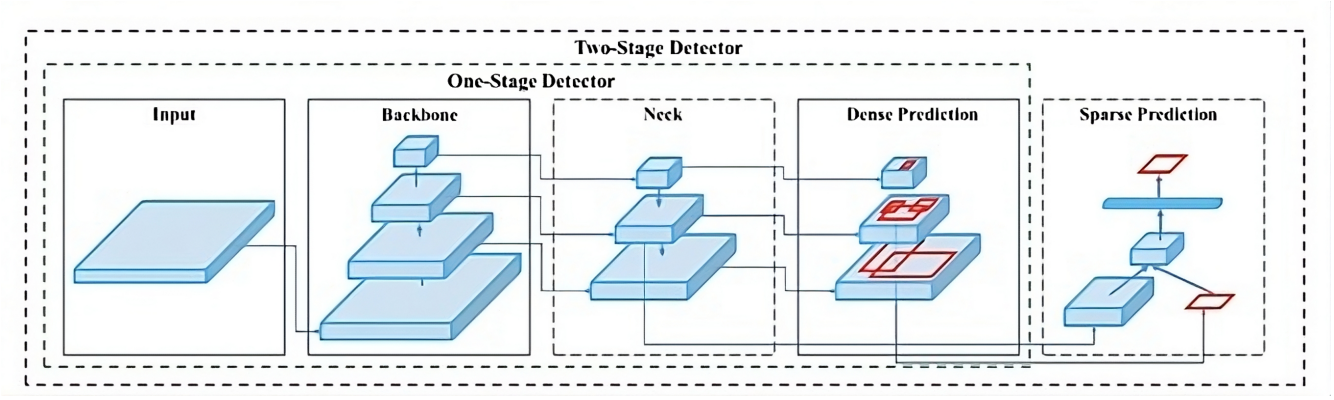

Figura 3: *Arquitectura del proceso de detección de objetos.*

mejores pesos (`best.pt`) y el modelo final (`last.pt`) para su posterior evaluación.

El modelo YOLOv11 fue entrenado utilizando la nueva interfaz de Ultralytics para modelos más recientes, también implementada en PyTorch. Esta versión experimental introduce mejoras en la arquitectura, atención y regularización, buscando una mayor capacidad de generalización en entornos complejos.

Se utilizó el siguiente fragmento de código en Python para entrenar el modelo:

```
from ultralytics import YOLO

# Cargar modelo preentrenado
model = YOLO("yolo11m.pt")

# Entrenamiento personalizado
train = model.train(
    data="./dataset-v11.yaml",
    epochs=25,
    imgsz=640,
    batch=2,
    device=0,
    project="../results",
    name="v3",
)
```

En este caso, se utilizó la variante `yolo11m.pt` como punto de partida. El número de épocas se fijó en 25, con un tamaño de imagen de 640 píxeles y un tamaño de lote reducido a 2, debido al mayor uso de memoria de esta arquitectura. El modelo fue entrenado en GPU (dispositivo 0) y los resultados se almacenaron en un proyecto específico para facilitar su análisis.

Ambos modelos fueron evaluados posteriormente con métricas estándar, lo que permitió identificar fortalezas y debilidades de cada versión en el contexto específico del proyecto.

4.1 Hiperparámetros

Los hiperparámetros definidos para realizar el entrenamiento fueron:

- lr0: 0.01
- lrf: 0.01
- momentum: 0.937
- weight_decay: 0.0005
- warmup_epochs: 3
- warmup_momentum: 0.8

Algunos otros fueron el umbral de Intersección sobre la Unión (IoU), indicado por el parámetro `iou_t`, establece un valor mínimo que determina cuándo una detección se considera correcta en función del grado de superposición con el objeto real. Un umbral bajo aumenta el número de detecciones, mientras que un valor más alto hace que el modelo sea más riguroso. El parámetro `fliplr` controla la probabilidad de voltear las imágenes horizontalmente durante la augmentación de datos, configurado aquí en 0.5 para que la mitad de las imágenes se voltee aleatoriamente. Por otro lado, la técnica `mosaic` de augmentación permite combinar varias imágenes en una sola, diversificando las variaciones en las escenas de entrenamiento y contribuyendo así a una mejora en la generalización del modelo. Finalmente, se emplearon aumentaciones adicionales mediante la biblioteca `albumentations`, aplicando de forma aleatoria desenfoques, conversiones a escala de grises, y ajustes de contraste, entre otros, para incrementar la robustez del modelo frente a distintas condiciones visuales en las imágenes. Para el proceso de optimización del modelo, se utilizó el optimizador SGD (Stochastic Gradient Descent) con una tasa de aprendizaje inicial (`lr`) de 0.01. El SGD es ampliamente utilizado en entrenamientos de deep learning por su efectividad en la

convergencia y control de los gradientes, especialmente en tareas de visión por computadora. Adicionalmente, se configuraron diferentes grupos de parámetros con distintos valores de `weight decay`. En este caso, los pesos sin decaimiento fueron 57, mientras que 60 pesos aplicaron un `weight decay` de 0.0005.

4.2 Métricas

Para evaluar el desempeño de los modelos, se utilizaron métricas ampliamente reconocidas en tareas de detección de objetos, como precisión (*precision*), *recall* y F1-score. Estas métricas permiten medir la eficacia del modelo en identificar correctamente la presencia de ganado en imágenes de prueba. En este contexto:

- **Precisión**: Indica la proporción de detecciones correctas (vacas identificadas) frente al total de detecciones realizadas.
- **Recall**: Mide la capacidad del modelo para encontrar todos los objetos de interés presentes en las imágenes.
- **mAP@0.5**: Esta métrica, conocida como *mean Average Precision* al 50 % de *IoU*, evalúa la precisión promedio del modelo considerando que las predicciones correctas deben alcanzar al menos un 50 % de solapamiento (IoU) con los objetos reales. Es una métrica comúnmente utilizada para evaluar la precisión en tareas de detección de objetos.
- **mAP@0.5:0.95**: Similar al mAP@0.5, esta métrica evalúa la precisión promedio del modelo, pero calcula el promedio de IoUs desde el 50 % hasta el 95 % en intervalos del 5 %. Este rango permite obtener una medida más exigente del desempeño del modelo, ya que evalúa su precisión en una gama más amplia de solapamientos posibles, proporcionando una visión más completa de la calidad de las predicciones.

5 Tuning

El proceso de refinamiento del modelo se dividió en tres iteraciones principales, donde cada iteración implementó ajustes específicos en los hiperparámetros, arquitectura y datos de entrenamiento con el objetivo de mejorar el desempeño general del modelo.

5.1 Iteraciones

A continuación, se describen las características principales de cada etapa en los modelos:

Modelo 1: YOLOv5 Vainilla y entrenamiento default

Inicialmente, se realizó un entrenamiento *vainilla* como primera iteración y se utilizó la arquitectura *YOLOv5* con hiperparámetros predeterminados para establecer una línea base de desempeño. Este modelo fue entrenado bajo los siguientes parámetros:

- **Arquitectura:** *YOLOv5*.
- **Pesos Iniciales:** *yolov5s.pt* (small).
- **Hiperparámetros:** Predeterminados en la configuración de YOLOv5.
- **Entrenamiento:** 25 épocas con un tamaño de lote de 4 imágenes.

Modelo 2: YOLOv5 Refinado + Transfer Learning

Durante esta iteración, se buscó realizar una mejora al modelo inicial, por lo que se realizó *Transfer Learning* al modelo de primera iteración, utilizando como pesos iniciales un modelo preentrenado para la detección de ganado en espacio de descanso (camas).

- **Arquitectura:** *YOLOv5*.
- **Hiperparámetros customizados:**
  1. **imgsz:** Se aumentó el *image size* a 1920 para capturar más detalles en las imágenes del dataset.
- **Pesos Iniciales:** *best.pt* (transfer learning).
- **Entrenamiento:** 25 épocas con un tamaño de lote de 2 imágenes.

Modelo 3: YOLOv11 Vainilla + Dataset Extendido

Finalmente, se integró el entrenamiento con arquitectura YOLOv11 para realizar una mejora adicional al segundo modelo. Sumado a esto, se integró un dataset previamente etiquetado, de vacas en espacio de descanso (camas) para brindar al modelo más información en imágenes para su aprendizaje. Gracias a las características que hacen más robusto a YOLOv11, y a la extensión del dataset original, se obtuvieron resultados muy favorables en desempeño.

- **Arquitectura:** *YOLOv11*.
- **Pesos Iniciales:** *yolo11m.pt* (medium).
- **Entrenamiento:** 25 épocas con un tamaño de lote de 2 imágenes.

6 Resultados

Los resultados y métricas obtenidas en validación por los tres modelos se pueden apreciar en la siguiente tabla.

	Modelo 1	Modelo 2	Modelo 3
Arquitectura	YOLOv5	YOLOv5	YOLOv11
Pesos iniciales	YOLOv5s	best.pt	YOLOv11m
Épocas	25	25	25
Conjunto de datos (número de imágenes)	4,213	4,213	8,163
Box Loss	0.0054	0.0022	0.4307
Object Loss	0.0004	0.0011	-
Precisión	66.90 %	77.67 %	97.23 %
Recall	31.61 %	84.00 %	96.89 %
mAP@0.5	47.39 %	80.61 %	99.03 %
mAP@0.5:0.95	27.74 %	52.32 %	82.63 %

Cuadro 1: *Comparación de métricas y resultados entre las iteraciones del modelo.*

6.1 Interpretación

Segundo Modelo

Las métricas de este modelo muestran una alta variabilidad, en especial en la precisión y en el mAP@0.5. Esto sugiere que el modelo aún está aprendiendo patrones, pero podría ser sensible a ciertas imágenes. No existe una convergencia estable en las métricas para este modelo, por lo que podemos concluir que está parcialmente ajustado.

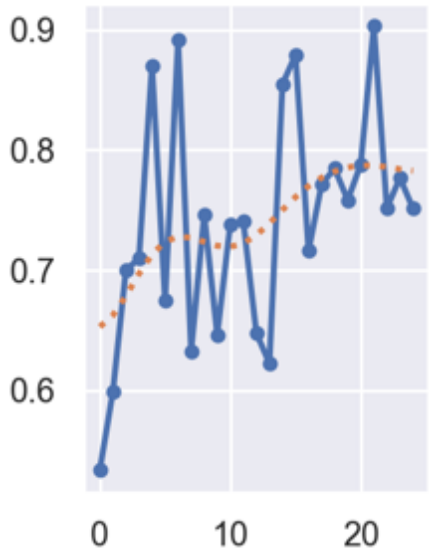

Figura 4: *Precisión del segundo modelo a lo largo del entrenamiento.*

La precisión muestra una tendencia general al alza con cierta variabilidad.

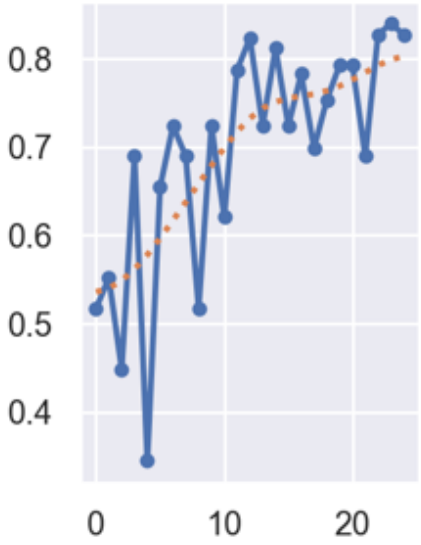

Figura 5: *Recall del segundo modelo a lo largo del entrenamiento.*

El recall presenta una mejora sostenida a lo largo del entrenamiento, comenzando cerca de 0.4 y superando el 0.8 hacia la época 20, esto implica que el modelo mejora progresivamente su capacidad para detectar todos los objetos relevantes (disminución de falsos negativos).

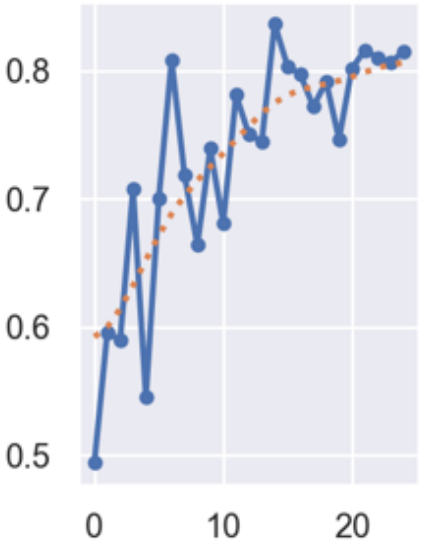

Figura 6: *Mean Average Precision @ 50 % del segundo modelo a lo largo del entrenamiento.*

El mAP@0.5 (mean Average Precision a IoU 0.5) inicia cerca de 0.5 y termina por encima de 0.8, con picos cercanos a 0.85.

Tercer Modelo

A diferencia del segundo modelo, las curvas de las métricas para el último modelo del proyecto son suaves, consistentes y no presentan oscilaciones abruptas. Las métricas se estabilizan en valores altos sin sobrepasar el 100 %, lo cual sugiere que el modelo aprendió adecuadamente de los datos sin memorizar casos específicos.

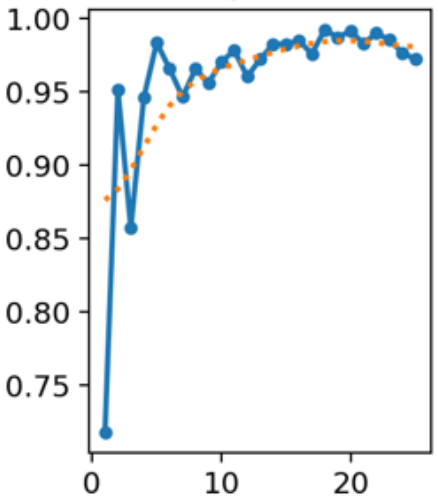

Figura 7: *Precisión del tercer modelo a lo largo del entrenamiento.*

La precisión inicia en 0.70 y rápidamente supera 0.95 en las primeras épocas, estabilizándose cerca de 0.98 - 0.99.

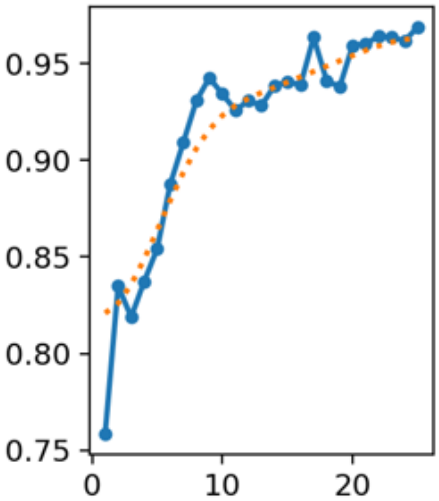

Figura 8: *Recall del tercer modelo a lo largo del entrenamiento.*

El recall empieza en 0.75 y mejora constantemente hasta superar 0.96 hacia el final del entrenamiento. Esto indica una capacidad muy alta del modelo para detectar la mayoría de las vacas en las imágenes (bajo índice de falsos negativos).

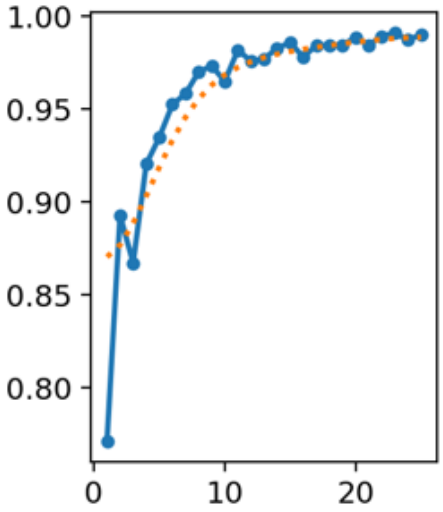

Figura 9: *Mean Average Precision @ 50 % del tercer modelo a lo largo del entrenamiento.*

La métrica mAP@0.5 comienza en 0.8 y llega a 0.99, lo que muestra una excelente capacidad del modelo para localizar y clasificar correctamente objetos cuando se permite cierta tolerancia en el matching.

Como conclusión, no hay evidencia de overfitting ni underfitting. El modelo está bien ajustado a los datos, posiblemente gracias al uso del dataset combinado y la estrategia de congelar capas para conservar generalización.

7 Detecciones

Para ilustrar los resultados del modelo, se desarrolló un script en Python que carga el modelo entrenado utilizando PyTorch, a través de la librería Ultralytics YOLO. Este script permite realizar predicciones sobre un conjunto de imágenes de prueba, evaluando así la capacidad del modelo para identificar la presencia de vacas en entornos reales, como potreros o establos.

Durante la ejecución del script, se procesan las imágenes y se generan visualizaciones que incluyen las predicciones del modelo, mostrando los cuadros delimitadores (bounding boxes) alrededor de las vacas detectadas. Además, se superponen etiquetas que indican la clase predicha y su respectivo nivel de confianza, expresado como un valor entre 0 y 1. Esta representación gráfica no solo permite verificar la efectividad del modelo, sino que también proporciona una comprensión visual clara del comportamiento del modelo ante

diferentes condiciones ambientales y posturas de los animales.

Adicionalmente, se implementó un segundo script orientado al seguimiento de vacas en secuencias de video. Este script procesa los fotogramas de un video y realiza inferencias cuadro por cuadro, generando anotaciones visuales solo en aquellos en los que se detecta al menos una vaca. Estas imágenes anotadas se almacenan en un directorio de salida, permitiendo evaluar el desempeño del modelo en escenarios más dinámicos y cercanos al mundo real. El enfoque de seguimiento es útil para futuras aplicaciones que requieran análisis temporal, como conteo de animales, monitoreo de comportamiento o generación de alertas en tiempo real.

Figura 10: *Imagen con anotaciones predichas por el segundo modelo entrenado.*

Figura 11: *Imagen con anotaciones predichas por el tercer modelo entrenado.*

8 Conclusiones

A lo largo de este proyecto, se logró desarrollar un sistema de detección de vacas utilizando modelos basados en la arquitectura YOLOv5, entrenados y afinados sobre un conjunto de datos específico. A través de distintas estrategias de entrenamiento —incluyendo el ajuste de hiperparámetros y la congelación parcial de capas— se alcanzaron métricas de desempeño destacadas, especialmente en el tercer modelo, que superó el 97 % de precisión y recall, y presentó una alta estabilidad en el aprendizaje.

Los resultados obtenidos demuestran que el modelo final no solo está bien ajustado a los datos, sino que también muestra una excelente capacidad de generalización, sin evidencia clara de overfitting ni underfitting. Esto se confirma tanto en los gráficos de evaluación como en las detecciones visuales sobre imágenes y secuencias de video reales.

El sistema desarrollado no solo permite identificar vacas de forma precisa, sino que sienta las bases para futuras aplicaciones en el sector agropecuario, como el monitoreo automatizado del comportamiento animal, el conteo inteligente de ganado o la generación de alertas en tiempo real. En conjunto, este proyecto representa una contribución significativa al uso de la visión por computadora en entornos rurales, facilitando la toma de decisiones informadas y mejorando la eficiencia operativa en el campo.

9 Bibliografía

[1] Ultralytics. *YOLOv11: The most recent YOLO Architecture Release*.
https://github.com/ultralytics/ultralytics

[2] Ultralytics. *YOLOv5: A State-of-the-Art Object Detection Architecture*.
https://github.com/ultralytics/yolov5

[3] Ultralytics. *Ultralytics Train Custom Data*.
`https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/`

[4] Ultralytics. *Comprehensive Guide to Ultralytics YOLOv5*.
https://docs.ultralytics.com/yolov5/

[5] Ultralytics. *Ultralytics YOLOv5 Architecture*.
`https://docs.ultralytics.com/yolov5/tutorials/architecture_description/`

[6] Khanam, Hussain. *YOLOv11: An Overview of the Key Architectural Enhancements*.
https://arxiv.org/pdf/2410.17725v1

[7] Bochkovskiy, Alexey, Wang, Chien-Yao, and Liao, Hong-Yuan Mark. *YOLOv4: Optimal Speed and Accuracy of Object Detection*.
https://arxiv.org/pdf/2004.10934

[8] Roboflow. *What is YOLOv5? A Guide for Beginners*. https://blog.roboflow.com/yolov5-improvements-and-evaluation/

[9] Tsang, Sh. *Reading: C3 — Concentrated-Comprehensive Convolution (Semantic Segmentation)*. https://sh-tsang.medium.com/reading-c3-concentrated-comprehensive-convolution-semantic-segmentation-5b6dd3fb46b2

[10] Tsang, Sh. *Brief Review: YOLOv5 for Object Detection*. https://sh-tsang.medium.com/brief-review-yolov5-for-object-detection-84cc6c6a0e3a

[11] Hui, Jonathan. *mAP (mean Average Precision) for Object Detection*. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173