

CARRERA DE ESPECIALIZACIÓN EN INTELIGENCIA ARTIFICIAL

MEMORIA DEL TRABAJO FINAL

Identificación de estados fenológicos de la flor de durazneros mediante visión por computadora

Autor:

Ing. Héctor Luis Sánchez Márquez

Director:

Ing. Juan Ignacio Cavalieri (FIUBA)

Codirector:

Esp. Lic. Nicolás Eduardo Horro (INVAP S.E.)

Jurados:

Esp. Lic. María Carina Roldán (FIUBA)

Esp. Ing. Alfonso Rafel (DeepAgro)

Esp. Ing. Ariadna Garmendia (FIUBA)

*Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires,
entre agosto de 2023 y agosto de 2024.*

Resumen

Esta memoria presenta un algoritmo desarrollado para el Instituto Nacional de Tecnología Agropecuaria que es capaz de identificar los estados fenológicos de la flor de duraznero y extraer información de su vareta a partir de imágenes. El objetivo de este desarrollo es agilizar y automatizar la toma de datos de las varetas de duraznero a través de fotos. Con ello, se busca aumentar el caudal de datos existente y conocer el estado fenológico a campo.

Para su desarrollo e implementación fueron aplicados los conocimientos de visión por computadora, análisis de datos, aprendizaje profundo y buenas prácticas de despliegue adquiridas en la carrera.

Agradecimientos

A mi familia, por siempre brindarme su apoyo incondicional en cada uno de mis pasos.

A Carolina, mi novia, con quien tuve innumerables catarsis y superé cada uno de los desafíos que este posgrado presentó.

A Adonis, mi amigo, sin él no hubiera descubierto mi pasión por el mundo de la inteligencia artificial.

A mis tutores, Juan Ignacio Cavalieri y Nicolás Eduardo Horro, quienes aceptaron guiarme y compartir su tiempo y conocimientos para la realización de este trabajo.

Índice general

Resumen	I
1. Introducción general	1
1.1. Descripción de la problemática	1
1.2. Motivación	2
1.3. Requerimientos	3
1.4. Objetivo y alcances	4
1.5. Estado del arte	4
1.5.1. Medición de la vareta	4
1.5.2. Detección de los estados fenológicos de la flor de duraznero	5
1.5.3. Conteo de flores	5
2. Introducción específica	7
2.1. Red neuronal convolucional	7
2.2. Detección de objetos	8
2.2.1. Detectores de dos etapas	9
Detección de objetos con <i>Fast R-CNN</i>	9
Detección de objetos con <i>Faster R-CNN</i>	9
2.2.2. Detectores de una etapa	10
Detección de objetos con <i>YOLO</i>	10
2.3. Clasificación de imágenes	11
3. Diseño e implementación	13
3.1. Arquitectura del sistema	13
3.2. Preparación de los datos	15
3.2.1. Análisis exploratorio	15
Balanceo del conjunto de datos antes del etiquetado	17
3.2.2. Etiquetado del conjunto de datos	17
Balanceo del conjunto de datos después del etiquetado	18
3.2.3. Aumento de los datos	20
3.2.4. Preprocesamiento de los datos	20
3.3. Módulo de estimación de longitud	21
3.4. Módulo de detección y procesamiento	22
3.4.1. Detección de estados fenológicos	22
3.4.2. Conteo de flores	23
3.4.3. Clasificador de flores	24
3.5. Integración de módulos y generación de resultados	26
3.6. Interfaz de usuario	27
4. Ensayos y resultados	31
4.1. Banco de pruebas	31
4.2. Desempeño de modelos	31
4.2.1. Desempeño del detector de regla	32

4.2.2.	Desempeño del detector de vareta	33
4.2.3.	Desempeño del detector de estados fenológicos	33
<i>Faster R-CNN</i> sin aumento de datos	34	
<i>Faster R-CNN</i> con aumento de datos	35	
<i>YOLOv8n</i> sin aumento de datos	35	
<i>YOLOv8n</i> con aumento de datos	37	
4.3.	Desempeño de módulos	38
4.3.1.	Desempeño del módulo de estimación de longitud	39
4.3.2.	Desempeño del módulo de detección y procesamiento	40
5.	Conclusiones	43
5.1.	Conclusiones generales	43
5.2.	Próximos pasos	44
Bibliografía		45

Índice de figuras

1.1.	Proceso para obtener los datos genómicos [1].	2
1.2.	Frutales afectados por las heladas primaverales [2].	2
2.1.	Ejemplo de arquitectura de red neuronal convolucional [9].	8
2.2.	Ejemplo de <i>template matching</i> para identificar el logo de Coca-Cola.	8
2.3.	Ejemplo de arquitectura de red neuronal de dos etapas <i>R-CNN</i> [13].	9
2.4.	Ejemplo de arquitectura de red neuronal de dos etapas <i>Faster R-CNN</i> [19].	10
3.1.	Arquitectura general del sistema.	13
3.2.	Módulo de estimación de longitud.	14
3.3.	Módulo de detección y procesamiento.	15
3.4.	Fotos de muestra del conjunto de datos.	16
3.5.	Fotos de varetas de duraznero con su tipo de flor.	17
3.6.	Fotos de estados fenológicos de la flor de duraznero.	17
3.7.	Desbalance de clases para el detector de estados fenológicos de la flor de duraznero.	19
3.8.	Desbalance de clases para el clasificador de flores de duraznero. . .	19
3.9.	Resultados del módulo de estimación de longitud.	22
3.10.	Imagen de salida del detector de estados fenológicos.	23
3.11.	Ejemplo de conteo de flores por vareta con regiones de interés. . . .	24
3.12.	Ejemplo de tabla resultante del módulo de estimación de longitud.	26
3.13.	Ejemplo de tabla resultante del módulo de clasificación y procesamiento.	26
3.14.	Ejemplo de tabla unificada.	26
3.15.	Pantalla de inicio del programa.	27
3.16.	Página para cargar imágenes.	28
3.17.	Barra de confianza para el módulo de estimación de longitud. . . .	29
3.18.	Barra de confianza para el módulo de detección y procesamiento. .	29
3.19.	Botones de descarga de resultados.	30
4.1.	Matriz de confusión normalizada de <i>YOLOv8n</i> para el conjunto de prueba.	37
4.2.	Mapa de calor con <i>Eigen-CAM</i> en la capa <i>C2f</i> del modelo <i>YOLOv8n</i> .	38
4.3.	Desempeño del módulo de estimación de longitud.	39
4.4.	Desempeño del módulo de detección y procesamiento con el uso de centroides con flores campanuláceas.	40

Índice de tablas

3.1. Características de las fotos de duraznos.	15
3.2. Cantidad de fotos etiquetadas por modelo.	18
3.3. Arquitectura de CNN para clasificación de flores de duraznero.	25
4.1. Métricas de detección para el detector de regla.	32
4.2. Métricas de detección para el detector de varetas sin aumento de datos.	33
4.3. Métricas de detección para el detector de varetas con aumento de datos.	33
4.4. Métricas de detección para <i>Faster R-CNN</i> sin aumento de datos.	34
4.5. Métricas de detección para <i>Faster R-CNN</i> con aumento de datos.	35
4.6. Métricas de detección para <i>YOLOv8n</i> sin aumento de datos.	36
4.7. Métricas de detección para <i>YOLOv8n</i> con aumento de datos.	37
4.8. Tabla de resultados del módulo de detección y procesamiento para la imagen de muestra.	40

Capítulo 1

Introducción general

En este capítulo se presenta la problemática y la motivación que llevaron a la realización del presente trabajo.

1.1. Descripción de la problemática

La fenómica hace referencia a la obtención de un gran caudal de datos de las características de las plantas, lo que se denomina el fenotipo de la planta. Esta disciplina está en auge en la actualidad debido a sus aplicaciones potenciales. Por un lado, habilita el mejoramiento a gran escala debido a que es necesario vincular una gran cantidad de datos genéticos con datos fenotípicos para identificar la función de los genes. Por otro lado, si se incluyen otros conjuntos de datos como son los climáticos, permite realizar predicciones precisas sobre el comportamiento de las variedades, lo que es necesario para implementar lo que se conoce como agricultura de precisión. Sin embargo, la fruticultura no ha dado el salto hacia la fenómica.

En la Estación Experimental Agropecuaria (EEA) de San Pedro se ha logrado secuenciar el ADN de más de 250 variedades de duraznero [1] y se dispone de una base de datos genómica de 75 gigabases (Gb) de ADN. Esta base permite identificar genes que controlan características del duraznero mediante algoritmos de inteligencia artificial (IA). Además, se dispone de datos climáticos diarios que se toman de forma automática que incluyen: las temperaturas medias, precipitaciones, horas de frío, radiación, etc. Esta información se combina con los datos genómicos y posteriormente, con modelos de IA, se predice el comportamiento de las variedades en escenarios climáticos futuros. En la figura 1.1 se observa el proceso para obtener los datos genómicos.

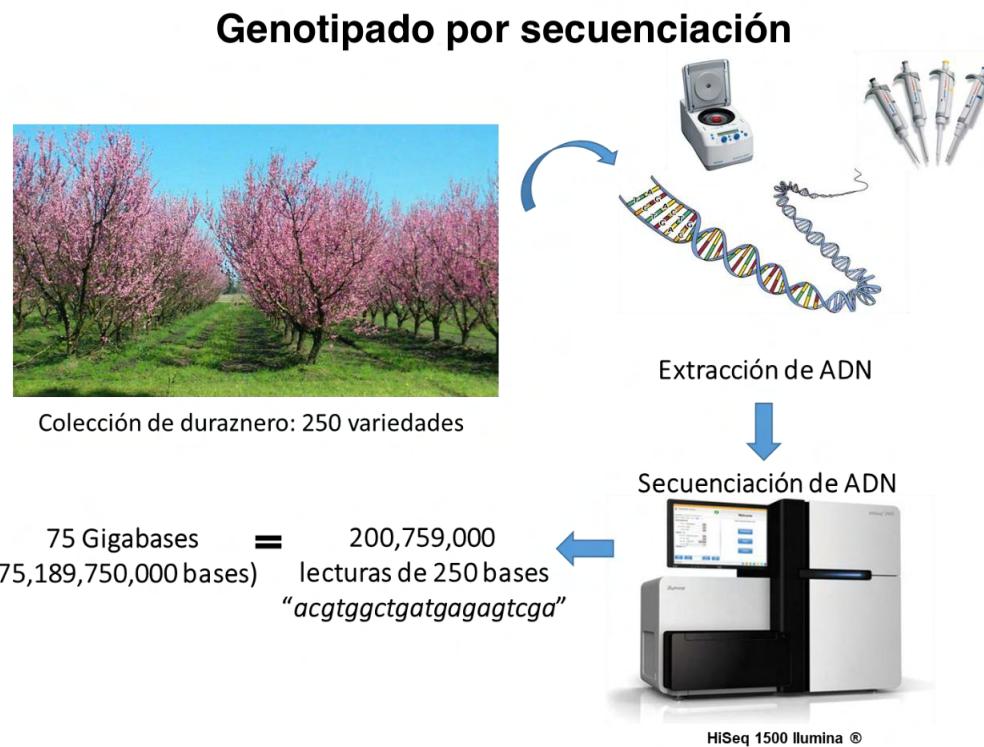


FIGURA 1.1. Proceso para obtener los datos genómicos [1].

En la actualidad, las heladas primaverales son el mayor problema de los frutales a nivel mundial. Este fenómeno ocurre cuando las flores abiertas se someten a temperaturas cercanas a los -2,5 °C. Las heladas primaverales tienen una temperatura parecida a cualquier otra helada que se puede presentar en la temporada de invierno. Sin embargo, estas heladas suelen presentarse después del invierno, creando un gran impacto contra las flores y los frutos. Los productores de frutas, en general, se ven altamente afectados porque deben pagar un alto precio por estas inesperadas heladas tardías. En la figura 1.2 se puede observar cómo este fenómeno meteorológico afecta a los frutales.



FIGURA 1.2. Frutales afectados por las heladas primaverales [2].

1.2. Motivación

Por lo explicado anteriormente, es del interés del Instituto Nacional de Tecnología Agropecuaria (INTA) determinar el estado fenológico a campo y mejorar la

tolerancia a heladas primaverales.

Para determinar el estado fenológico a campo, es necesario conocer el número de flores que se encuentran en estado vulnerable ante un pronóstico de heladas primaverales, así como también la densidad de flores.

En cuanto al mejoramiento, se ha realizado una caracterización a gran escala de la tolerancia a heladas de la colección de duraznero con el objetivo de identificar los genes responsables. Parte de ese experimento consistió en registrar el estado fenológico mediante fotos.

El presente trabajo permitirá automatizar la toma de datos de varetas de duraznero a partir de fotos para aumentar el caudal de datos y mejorar los modelos de IA.

1.3. Requerimientos

1. Requerimientos funcionales

- a) El sistema tomará como entrada imágenes de varetas de durazneros en formato JPG.
- b) El algoritmo debe detectar la presencia de las varetas de los durazneros e identificar el tipo de flor que posee.
- c) El algoritmo debe identificar el estado fenológico de cada flor de duraznero en la vareta. Este estado se clasificará como flor abierta, flor cerrada, flor sinpetalos, incierto.
- d) El algoritmo debe determinar la cantidad de flores por centímetro de vareta.
- e) El sistema debe entregar como resultado un archivo en formato CSV con los datos detectados por el algoritmo y una imagen donde se puedan visualizar las detecciones.
- f) El sistema debe funcionar en una computadora local.

2. Requerimientos de diseño e implementación

- a) El diseño debe ser modular.
- b) El algoritmo se elaborará en una *notebook* de Google Colab, utilizando el lenguaje de programación Python y bibliotecas de IA correspondientes.

3. Requerimiento de evaluación y prueba

- a) El modelo se evaluará con imágenes provenientes del mismo dataset de imágenes entregado por el cliente.
- b) La métrica que se utilizará para la evaluación del modelo de detección será *mean average precision* (mAP) y para el clasificador se tomarán en cuenta las métricas *accuracy*, *precision* y *recall*.

4. Requerimientos de documentación

- a) El funcionamiento del sistema debe estar correctamente explicado y documentado.

- b) El código estará correctamente comentado como parte de buenas prácticas del desarrollo de software.
- c) Inclusión de documentación en un repositorio, mediante un archivo README.md (opcional).

1.4. Objetivo y alcances

El objetivo de este trabajo es desarrollar un algoritmo que permita automatizar la toma de datos de las flores de duraznero a través de fotos de varetas.

El presente trabajo incluye:

- El preprocessamiento de las fotos para entrenar el modelo.
- La selección del modelo a entrenar.
- La elaboración de la *notebook* de pruebas en Python.
- La implementación local del modelo.

El trabajo no incluye:

- La recolección de datos/fotos.
- La integración con otros modelos que utilice el cliente.

1.5. Estado del arte

El presente trabajo contiene distintos algoritmos que, integrados, logran tomar los datos deseados. Es por este motivo que para determinar el estado del arte es necesario desglosar cada algoritmo y evaluarlo individualmente como se hace a continuación.

1.5.1. Medición de la vareta

En la actualidad, se han desarrollado algoritmos que pueden determinar el tamaño de distintos objetos a través de imágenes usando visión por computadora. Muchos parten de encontrar un objeto de referencia al que se le conocen sus dimensiones (alto y ancho). Este objeto de referencia, normalmente se selecciona por ser fácil de detectar, por conocer sus dimensiones y por ser un objeto único. El procedimiento habitual para su detección, es pasar la imagen a escala de grises, aplicar filtros gaussianos para eliminar el ruido, utilizar detección de bordes y por último utilizar detección de contornos, tal y como se realiza en el trabajo [3]. Cabe destacar que usualmente el fondo es de un color blanco, lo que facilita la detección.

En el presente trabajo se tienen imágenes con fondos de color naranja en su mayoría, el objeto de referencia a veces se encuentra oculto, las imágenes se encuentran en horizontal o vertical, el objeto de referencia no siempre tiene la misma posición, etc. Por estos motivos, se utilizó un método de detección más complejo con un modelo de detección de objetos que se conoce como YOLOv8 y se considera el estado del arte a la fecha.

1.5.2. Detección de los estados fenológicos de la flor de duraznero

La detección de flores ha sido estudiada con diferentes enfoques y arquitecturas de *deep learning*, como por ejemplo el estudio [4] que exploró la viabilidad de detección de estados fenológicos de las rosas con técnicas del contraste del color y comparando con el modelo de detección *Faster R-CNN*. Sin embargo, no utilizó ninguna arquitectura de una etapa para la detección, lo que podría ser más eficiente. Por otro lado, se tienen trabajos que sí utilizaron la arquitectura de una etapa, en específico de *YOLO* en sus versiones 4 y 5 como se presenta en [5] [6], pero su enfoque fue basado para las flores de kiwi.

La presente memoria busca en particular los estados fenológicos de la flor de duraznero utilizando y comparando dos modelos de detección, donde el primero tiene una arquitectura de dos etapas y el segundo tiene una arquitectura de una etapa. Este último corresponde a una arquitectura *YOLO* en su versión 8, que se propuso para la realización efectiva y eficiente de esta tarea, además de representar el estado del arte en la actualidad.

1.5.3. Conteo de flores

El conteo de objetos en imágenes a través de visión por computadora es otro campo que ha sido altamente estudiado y tiene muchos enfoques tanto simples como complejos. De esta forma, se expone el caso [7] donde se hace uso de redes convolucionales para predecir mapas de densidad que permiten hacer un conteo preciso de objetos en imágenes de alta densidad. Así como también se tiene el trabajo [8] que utiliza *transformers* con el mismo propósito.

Por otro lado, este trabajo propone una combinación entre el uso del modelo de detección de objetos y métodos de visión por computadora tradicionales para el conteo de las flores de duraznero.

Capítulo 2

Introducción específica

En este capítulo se presenta una introducción teórica detallada de los algoritmos utilizados para la elaboración del presente trabajo.

2.1. Red neuronal convolucional

Una red neuronal convolucional es un tipo de red neuronal diseñada para identificar patrones en imágenes. Particularmente esta arquitectura de *deep learning*, suele ser utilizada en problemas de visión por computadora relacionados a la clasificación o detección de objetos en una imagen. Estas redes, pueden llegar a tener cientos de capas y suelen estar compuestas por tres tipos principales como son las capas convolucionales, las capas de agrupación (*pooling*) y una o varias capas totalmente conectadas (*fully connected*) [9][10][11].

La capa convolucional consiste en un conjunto de filtros o *kernels* entrenables que se mueven por el ancho y alto de la imagen de entrada y calculan el producto escalar entre los píxeles de entrada y el filtro en cualquier posición. El resultado de este cálculo se incorpora en una matriz de salida. De esta forma, se desplaza el filtro repitiendo la operación anterior hasta que el *kernel* recorre toda la imagen. La serie de productos escalares de la imagen de entrada con los filtros se conoce como mapa de activación. Finalmente, la red aprenderá filtros que se activan cuando detectan algún tipo de característica, borde o patrón visual [10][11].

La capa de agrupación, se encuentra normalmente entre capas sucesivas convolucionales. Su función es simplificar la salida mediante la reducción no lineal de la tasa de muestreo, lo que termina resultando en una disminución de la cantidad de parámetros que la red debe aprender. Aunque se pierde información en esta capa, tiene beneficios para la red convolucional, como por ejemplo se reduce la complejidad, mejora la eficiencia y evita el sobreajuste [10][11].

La capa totalmente conectada, a diferencia de las otras capas mencionadas, tiene todos los nodos de la salida conectados directamente a los nodos anteriores. El objetivo de esta última capa es realizar una clasificación, basándose en las características extraídas anteriormente. La función de activación generalmente utilizada en esta capa es la función *softmax* para obtener la probabilidad de que un objeto pertenezca a una clase u otra entre un intervalo de 0 a 1.

En la figura 2.1 se puede observar un ejemplo de cómo se estructuran dichas capas en la red neuronal convolucional.

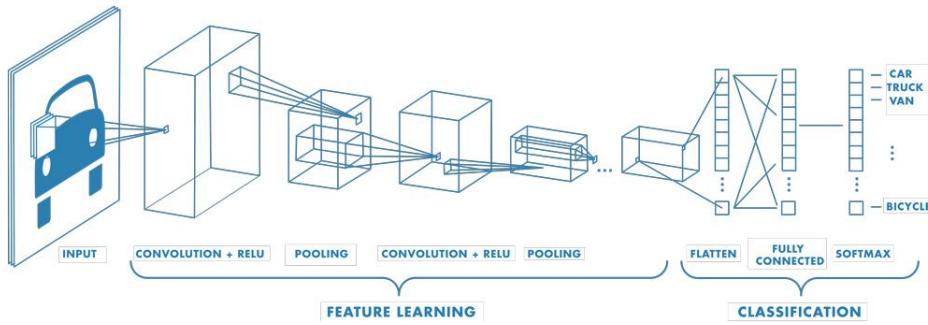


FIGURA 2.1. Ejemplo de arquitectura de red neuronal convolucional [9].

2.2. Detección de objetos

La detección de objetos es una técnica utilizada en visión por computadora para localizar e identificar uno o varios objetos en una imagen o vídeo. A diferencia de otras técnicas de *machine learning*, como es el caso de la clasificación o reconocimiento de imágenes, la detección busca localizar el lugar exacto donde se encuentra el objeto de interés y lo delimita con un rectángulo también llamado caja delimitadora o *bounding box* por su término en inglés. Por otro lado, una vez delimitado, el objeto se clasifica entre las categorías disponibles [12].

La detección de objetos se puede implementar utilizando métodos de *machine learning* clásicos o de *deep learning*, dependiendo del problema a resolver [13]. El uso de *deep learning* es más eficaz cuando se requiere tratar imágenes con muchas etiquetas, es decir, muchos objetos a detectar y que además tienen otras variaciones como cambio de brillo, rotaciones, cambio de escala, etc. Por otro lado, el uso de *machine learning* clásico es favorable cuando no se tienen altas capacidades de procesamiento y el número de etiquetas distintas a identificar es menor.

Algunos ejemplos de detección de objetos con *machine learning* clásico son SIFT [14], *template matching*, etc. En la figura 2.2 se puede observar el uso de *template matching* para detectar el logo de Coca-Cola en una imagen.



FIGURA 2.2. Ejemplo de *template matching* para identificar el logo de Coca-Cola.

Por otro lado, se tienen métodos de *deep learning* para detección de objetos más avanzados que involucran detectores de dos etapas y de una etapa como son *R-CNN*, *Faster R-CNN*, *SSD*, *YOLO*, entre otros.

2.2.1. Detectores de dos etapas

Los detectores de dos etapas como *R-CNN* y sus variantes, primero extraen las características de la imagen y luego proponen una región de interés (ROI). El ROI, consiste en un *bounding box* donde se presume que se encuentra el objeto a buscar. Luego, en la segunda etapa, se analizan las características encontradas en conjunto con el ROI, para seleccionar los *bounding boxes* finales y calcular las probabilidades de que el objeto en las regiones pertenezca a una clase específica [15].

Las redes neuronales de dos etapas son muy precisas al momento de detectar un objeto, sin embargo, son consideradas lentas durante la inferencia. Esta desventaja llevó a mejorar los modelos iniciales como *R-CNN* a sus variantes *Fast R-CNN* y *Faster R-CNN*.

En la figura 2.3 se puede observar la arquitectura de una red neuronal de dos etapas *R-CNN*.

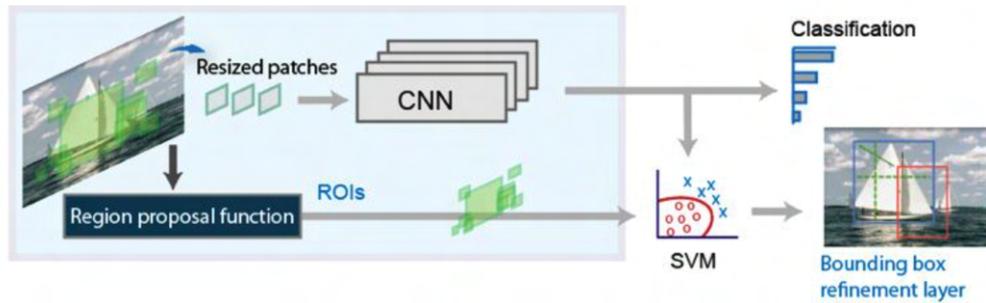


FIGURA 2.3. Ejemplo de arquitectura de red neuronal de dos etapas *R-CNN* [13].

Detección de objetos con *Fast R-CNN*

Las mejoras adicionadas en *Fast R-CNN* en comparación con *R-CNN* incluyen una nueva capa llamada *ROI Pooling*, que se encarga de extraer vectores de características de igual longitud de todas las regiones de interés propuestas. Por otro lado, *R-CNN* contiene tres etapas como son la generación de región propuesta, la extracción de características y la clasificación usando *SVM* [16], mientras que *Fast R-CNN* crea una red neuronal que tiene una única etapa, reduciendo así el número de etapas de su predecesor. Además, este modelo comparte cálculos computacionales a través de todas las ROIs propuestas en vez de hacerlo una a una de manera independiente. Por último, *Fast R-CNN* no guarda en caché las características, lo que reduce el uso de disco de memoria [17].

Detección de objetos con *Faster R-CNN*

El modelo *Faster R-CNN* fue diseñado para superar muchos de los errores encontrados en sus predecesores *Fast R-CNN* y *R-CNN*. En general, como su nombre en inglés sugiere, sus mejoras van relacionadas a la rapidez en comparación a las otras variantes.

Faster R-CNN introduce mejoras a través de la incorporación de la red de región propuesta o por sus siglas en inglés RPN, que es una red completamente convolucional que produce propuestas con diferentes escalas y relaciones de aspecto.

La RPN aplica la terminología de redes neuronales con atención para indicar al modelo dónde mirar.

En *Faster R-CNN* se introduce el concepto de *anchor boxes*, lo que permite detectar objetos en distintas escalas y relaciones de aspecto. Por último, se comparten cálculos computacionales a través de la RPN y *Fast R-CNN*, lo que reduce el tiempo computacional [18]. En la figura 2.4 se muestra un ejemplo de la arquitectura de *Faster R-CNN*.

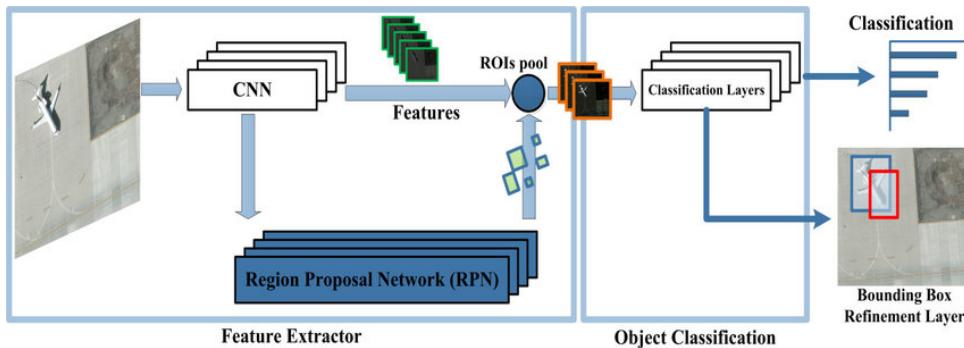


FIGURA 2.4. Ejemplo de arquitectura de red neuronal de dos etapas *Faster R-CNN* [19].

2.2.2. Detectores de una etapa

Los detectores de una etapa son modelos que omiten la etapa de RPN y ejecutan la detección directamente en un muestreo denso de ubicaciones. Estos modelos hacen uso de *grid-box* y *anchor box* para localizar la región de detección y delimitar al objeto de interés. En general, se consideran más rápidos que los detectores de dos etapas, sin embargo, son menos precisos [20].

Algunos ejemplos de detectores de una etapa son las arquitecturas *YOLO* [21], *SSD* [22], *RetinaNet* [23] y *SqueezeDet* [24].

Detección de objetos con YOLO

You Only Look Once (*YOLO*) es considerado el estado del arte para la detección en tiempo real y fue introducido en el año 2015. En el paper [21], los autores trataron la detección de objetos como un problema de regresión simple en vez de hacerlo como un problema de clasificación, en el que se separan espacialmente los *bounding boxes* y se les asocia una probabilidad a cada detección usando una red neuronal convolucional.

El algoritmo se basa en el uso de bloques residuales, *bounding boxes* por regresión, *Intersection Over Union* (IoU) y *Non-Maximum Suppression*.

El primer paso es el uso de bloques residuales que consiste en dividir la imagen original en celdas cuadriculadas de NxN. Cada celda en la cuadrícula es responsable de localizar y predecir la clase de objeto que cubre con su respectivo valor de confianza. Luego, se determinan los *bounding boxes* con los atributos que *YOLO* obtiene usando un módulo de regresión lineal. La respuesta que se recibe de este módulo es una representación vectorial de las coordenadas para cada *bounding box*.

Por otro lado, un mismo objeto puede tener múltiples detecciones durante la predicción y no todas serán relevantes. Por este motivo, se usa *Intersection Over Union* que ayuda a descartar aquellas predicciones no relevantes, dándoles un valor entre 0 y 1. En este proceso, se establece un umbral para el IoU, que posteriormente, se compara con el cálculo generado por YOLO para cada cuadrícula y si el resultado es menor al umbral establecido, se descarta dicha predicción. Por último, se aplica *Non-Maximum Suppression* para preservar solo aquellas predicciones que tengan un valor de confianza alto [25].

YOLO cuenta con varias versiones desde su lanzamiento donde se ha mejorado su velocidad, precisión y el uso de recursos. Estas versiones van desde la uno a la nueve, siendo la ocho la usada en la presente memoria.

2.3. Clasificación de imágenes

La clasificación de imágenes es una tarea fundamental en el ámbito de la visión por computadora, que consiste en asignar etiquetas o clases a las imágenes. Este proceso implica trabajar a nivel de píxeles, donde se extraen características de la imagen en formato vectorial, que luego son utilizadas por modelos de *machine learning* para realizar predicciones.

La clasificación puede llevarse a cabo mediante métodos de *machine learning* clásicos o utilizando *deep learning*. El uso de *deep learning* para esta tarea ha demostrado un rendimiento sobresaliente, incluso ante imágenes con cambios de iluminación, rotación, deformación, oclusión, entre otros desafíos.

La clasificación de imágenes mediante *deep learning* emplea redes neuronales convolucionales, un proceso que inicia con una imagen que pasa por una capa de entrada, donde se realiza un preprocesamiento antes de ser enviada a una capa convolucional. En la capa convolucional, se aplican filtros a la imagen para extraer características y generar un mapa de características que indica la presencia o ausencia de ciertos atributos. Posteriormente, la imagen se somete a una reducción de dimensionalidad en una capa de agrupación, de esta forma se preserva la información importante extraída, antes de pasar por una capa de activación no lineal, generalmente una capa ReLU. Finalmente, el resultado se alimenta a una capa totalmente conectada para obtener las probabilidades de que la imagen pertenezca a una de las clases predefinidas.

Capítulo 3

Diseño e implementación

En este capítulo se describe el diseño e implementación del sistema desarrollado que automatiza la toma de datos de las flores de duraznero a través de fotos de varetas.

3.1. Arquitectura del sistema

El sistema general implementado en este trabajo se presenta en la figura 3.1. Este sistema cuenta con un *frontend* donde el usuario sube una foto de vareta de duraznero que posteriormente pasará por dos módulos de forma secuencial. El primero se ocupa de medir la longitud de la vareta y el segundo módulo se encarga de detectar el estado fenológico, la cantidad y el tipo de flores que posee la vareta. Cada uno de los resultados obtenidos por estos módulos es presentado por pantalla al usuario que tiene la posibilidad de descargarlos en formato CSV.

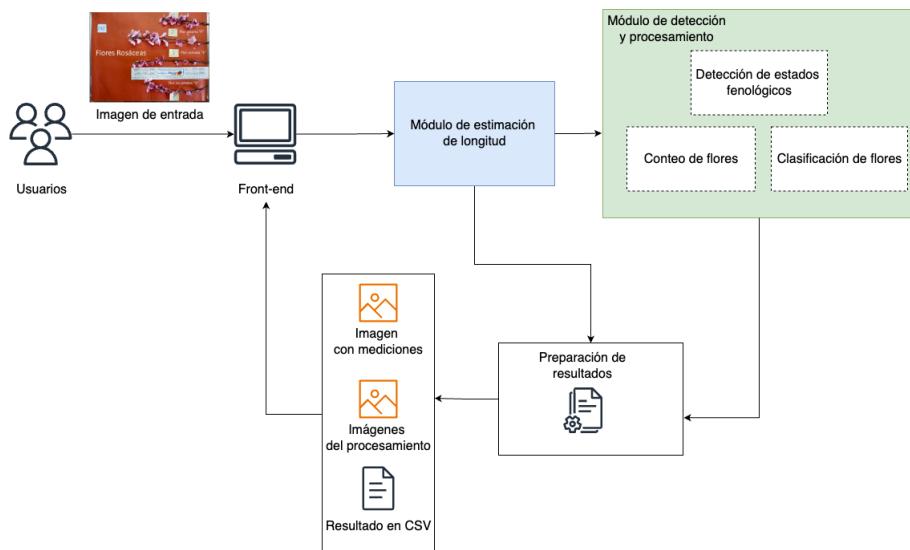


FIGURA 3.1. Arquitectura general del sistema.

La arquitectura del módulo que estima la longitud de la vareta se puede observar en la figura 3.2. Este módulo toma y preprocesa la imagen de entrada, posteriormente detecta la regla que es el objeto de referencia y procede a tomar las mediciones en píxeles de dicho elemento. Luego, se hace una conversión de píxeles a centímetros. Una vez finalizada la conversión, se detectan las varetas presentes en la imagen y se calculan sus dimensiones, se revisa la orientación de la foto y se toma la longitud de cada vareta en píxeles. Finalmente, con la conversión anterior

se pasa a centímetros las mediciones de las varetas, se anotan los resultados en una tabla y son enviados al siguiente módulo. Por otro lado, la imagen con las mediciones se muestra por pantalla.

Este módulo se detalla con más profundidad en este mismo capítulo.

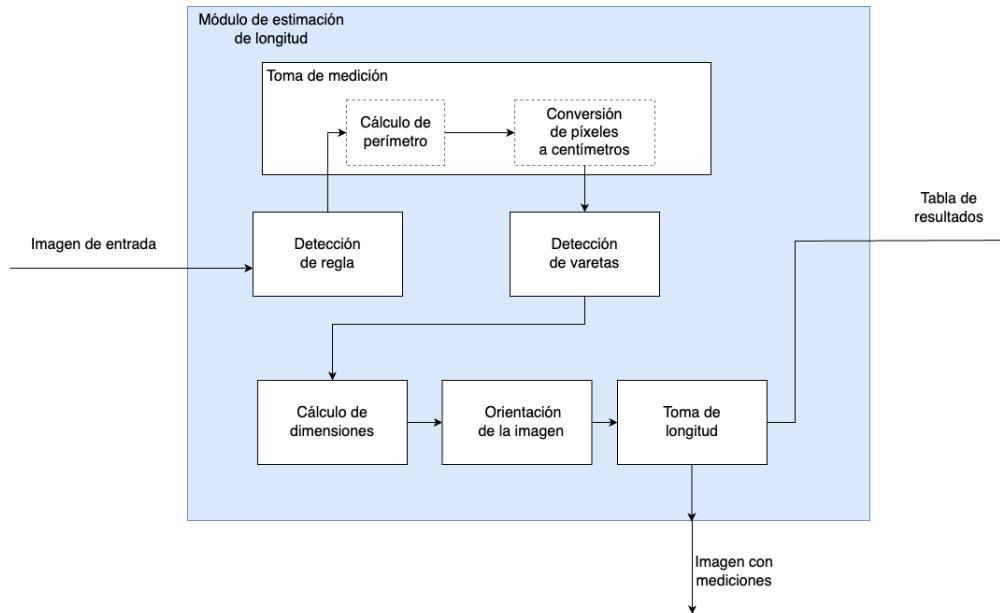


FIGURA 3.2. Módulo de estimación de longitud.

La arquitectura del módulo de detección y procesamiento se presenta en la figura 3.3. La entrada a este módulo es la imagen original donde se le aplica un preprocesamiento y posteriormente se pasa por un modelo que detecta los estados fenológicos de las flores de duraznero y sus varetas. Luego, se realiza un postprocesamiento de los resultados para conocer el número de flores totales que se encuentran en la imagen y la cantidad por vareta. Por último, se extraen las flores detectadas por vareta y son enviadas al clasificador para determinar su tipo.

Más adelante en este mismo capítulo se dará más información detallada de este módulo y de otros bloques del sistema que cumplen una función importante para generar los resultados deseados.

Cabe destacar que la realización de este sistema se llevó a cabo en el lenguaje de programación Python y se encuentra diseñado para funcionar en una computadora local bajo un ambiente virtualizado.

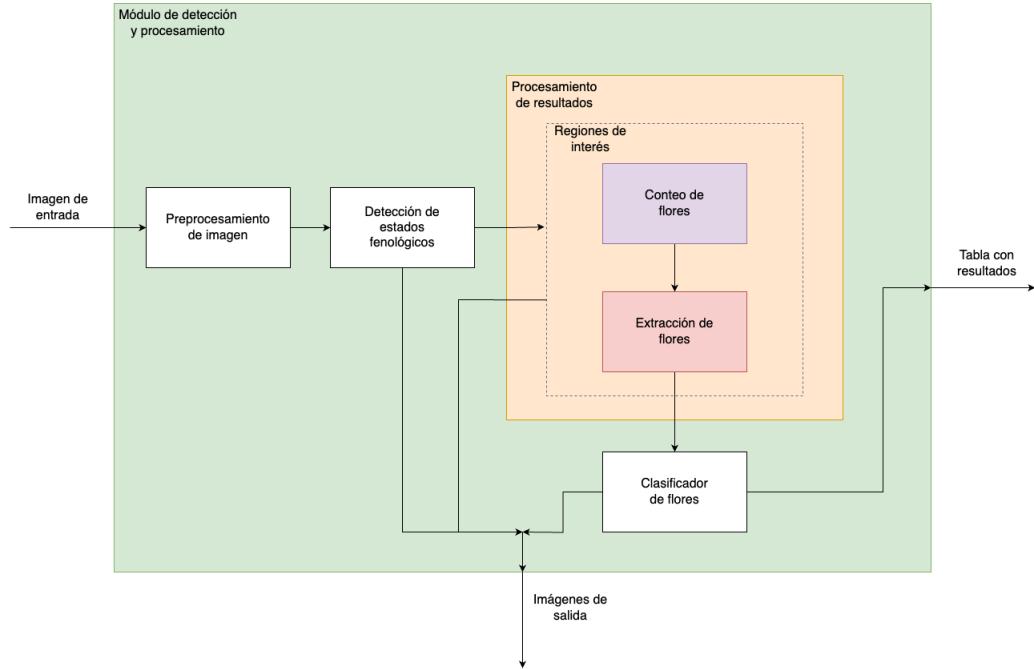


FIGURA 3.3. Módulo de detección y procesamiento.

3.2. Preparación de los datos

La preparación de los datos es una de las partes más críticas del desarrollo de un modelo de aprendizaje. Esto se debe a que si los datos no se encuentran en las condiciones adecuadas para su uso en un modelo de IA, el modelo no tendrá un buen rendimiento.

El presente trabajo requirió la generación de distintos etiquetados para el conjunto de datos original, por la cantidad de modelos utilizados durante su desarrollo.

3.2.1. Análisis exploratorio

El conjunto de datos original provisto por el INTA contiene las características que se indican en la tabla 3.1. En general, las fotos presentan distintas dimensiones y orientaciones (vertical u horizontal). Por otro lado, se desconoce el tipo de cámara utilizada y el ángulo siempre es cenital.

TABLA 3.1. Características de las fotos de duraznos.

Formato	Cantidad	Resolución	Observaciones
JPG	286	Variable	Cuatro varetas por foto.

Las imágenes tienen un fondo naranja y en ocasiones contienen partes grises. Esto corresponde a que las varetas de duraznero fueron posadas sobre una cartulina color naranja y a veces por las dimensiones de las varetas, se toma parte de la mesa de color gris.

Por otro lado, las fotos presentan cambios de brillo e iluminación, generando distintas tonalidades de los colores de fondo y de las flores.

Los elementos que se observan generalmente en las fotos provistas por el cliente incluyen: una regla, cuatro varetas, una cartulina de fondo, cuatro etiquetas que identifican las varetas y en ocasiones objetos que no forman parte de la detección o que no aportan información.

En la figura 3.4, se pueden observar algunos ejemplos de fotos con las características que se describieron anteriormente.

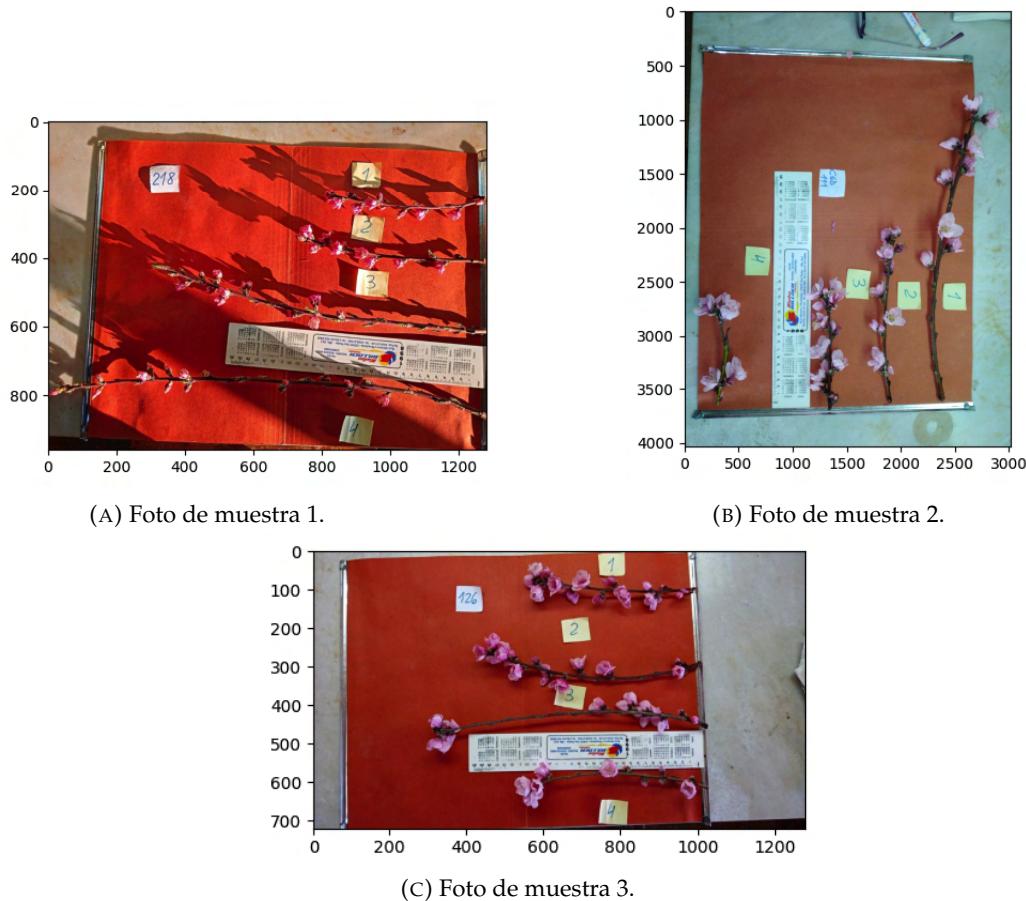


FIGURA 3.4. Fotos de muestra del conjunto de datos.

Es importante mencionar que las fotos de varetas tomadas por el cliente, están clasificadas por el tipo de flor que poseen las varetas. Es decir, una foto solo va a contener varetas con flores del tipo rosáceas o campanuláceas, pero no ambos tipos de flor en una misma imagen. En la figura 3.5 se muestran dos fotos de varetas, una que contiene flores rosáceas y otra que contiene flores campanuláceas.



(A) Foto con flores rosáceas.

(B) Foto con flores campanuláceas.

FIGURA 3.5. Fotos de varetas de duraznero con su tipo de flor.

Cabe destacar que en una foto de varetas de duraznero, se encuentran varias flores y cada flor posee un estado fenológico. Los estados fenológicos que se pueden encontrar son: flor abierta, flor cerrada y flor sin pétalos respectivamente. En la figura 3.6 se observan dichos estados fenológicos para la flor de duraznero del tipo rosácea.



(A) Flor abierta.

(B) Flor cerrada.

(C) Flor sin pétalos.

FIGURA 3.6. Fotos de estados fenológicos de la flor de duraznero.

Balanceo del conjunto de datos antes del etiquetado

El conjunto de datos cuenta con un total de 286 imágenes en total, donde 57 contienen varetas con flores del tipo campanulácea y las otras 229 con flores del tipo rosácea. Por lo tanto, se considera que el conjunto de datos se encuentra desbalanceado con respecto al tipo de flor.

3.2.2. Etiquetado del conjunto de datos

El conjunto de datos proporcionado por el cliente fue etiquetado utilizando una herramienta llamada *Roboflow* [26]. El etiquetado con *Roboflow* es simple y se puede utilizar para múltiples tareas relacionadas a visión por computadora. Por otro lado, el presente trabajo requirió varios etiquetados del mismo conjunto de datos, debido a los distintos modelos que se entrenaron para su realización.

Todos los modelos utilizados exceptuando el clasificador, utilizaron un etiquetado del tipo de detección de objetos, donde se delimitan los objetos de interés con *bounding boxes*. El etiquetado por modelo se detalla en la siguiente lista:

1. Detección de estados fenológicos: las etiquetas para el entrenamiento de este modelo fueron cinco y representan cada estado fenológico de la flor de duraznero. Estas etiquetas son: flor abierta, flor cerrada, flor sin pétalos, incierto y vareta. El tipo de etiquetado fue de detección de objetos.

2. Detección de regla: para este etiquetado se utilizó el mismo *dataset*, pero solo se tomó en cuenta la regla, por lo tanto, solo cuenta con esta única etiqueta para detección de objetos.
3. Detección de vareta: solo se etiquetaron las varetas, el motivo por el que se vuelve a hacer un etiquetado para este objeto se explica en secciones más adelante.

El conjunto de datos utilizado para entrenar el clasificador de flores se generó y etiquetó mediante un algoritmo que emplea el modelo que detecta los estados fenológicos, y posteriormente recorta cada flor detectada. Luego para distinguir entre las flores rosáceas y campanuláceas, se procesaron imágenes de ejemplares de cada tipo por separado, almacenando los resultados en archivos distintos clasificados por categoría.

Finalmente, por motivos de tiempo no se lograron etiquetar las 286 imágenes para cada caso. Los detalles de la cantidad de imágenes etiquetadas por conjunto de datos se encuentra en la tabla 3.2.

TABLA 3.2. Cantidad de fotos etiquetadas por modelo.

Modelo	Cantidad de imágenes etiquetadas
Detector de estados fenológicos.	140
Detector de regla.	131
Detector de vareta.	101

Balanceo del conjunto de datos después del etiquetado

El conjunto de datos para el detector de los estados fenológicos de la flor tenía un desbalance que se puede observar en la figura 3.7. La clase mayoritaria fue la correspondiente a flor abierta que es el estado fenológico con más presencia en las 140 fotos etiquetadas. Con esta distribución se infirió que los modelos tendrían un sesgo hacia la clase flor abierta y que la categoría que sería más difícil de detectar sería la clase incierto. La solución a este problema de desbalance para el detector de estados fenológicos se explica más delante en la presente memoria.

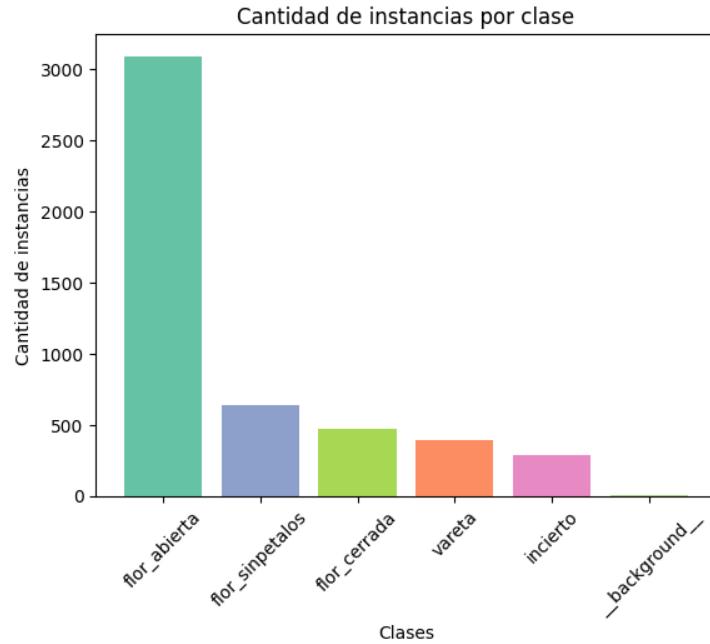


FIGURA 3.7. Desbalance de clases para el detector de estados fenológicos de la flor de duraznero.

Por otro lado, el conjunto de datos para el clasificador de flores contaba con un desbalance entre la cantidad de flores rosáceas y campanuláceas. Este desbalance se observa en la figura 3.8. La consecuencia prevista a esta desproporción es un posible sesgo hacia la clase mayoritaria. Más adelante se detallarán las medidas para mitigar este problema.

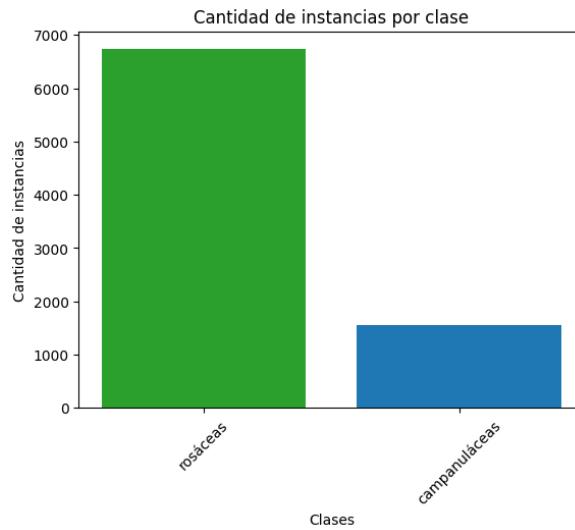


FIGURA 3.8. Desbalance de clases para el clasificador de flores de duraznero.

3.2.3. Aumento de los datos

Se utilizó *data augmentation* [27] para incrementar la cantidad de imágenes para el conjunto de datos destinados al modelo de detección de los estados fenológicos. Además las transformaciones de imágenes aplicadas buscan que el modelo sea capaz de generalizar mejor a pesar de encontrar cambios de brillo, iluminación, rotaciones, etc. Por otro lado, también se hizo uso de esta técnica para intentar balancear el conjunto de datos destinado al clasificador de flores.

Para el detector de estados fenológicos de las flores de duraznero se implementaron las siguientes transformaciones:

1. 90 grados de rotación en sentido horario y anti horario.
2. Cambio de brillo entre un -25 % y 25 %.
3. Desenfoque de 1 píxel.
4. Cambio de brillo por *bounding box*.

El total de imágenes aumentó a 336. Sin embargo, es de tener en cuenta que el número de fotografías disponible sigue siendo inferior al recomendado para el entrenamiento y validación de los distintos modelos de aprendizaje utilizados en la presente memoria. Por otro lado, con esta técnica no se pudo resolver el desbalance de datos observado en la figura 3.8, debido a que no hay una transformación directa de la imagen que permita la eliminación de las clases mayoritarias presentes en la detección. Más adelante, en la presente memoria, se explica el método implementado para mitigar el desbalanceo en este caso.

El detector de vareta también contó con un aumento de los datos para mejorar el rendimiento del modelo de detección utilizado y para incrementar la cantidad de muestras a un total de 243 imágenes. Las transformaciones utilizadas fueron:

1. Rotación horizontal.
2. Cambio de brillo entre un -15 % y 15 %.
3. Desenfoque de 1 píxel.
4. Ruido hasta un 0,1 % en píxeles.

Para el clasificador de flores se aplicaron transformaciones que involucraban giros o rotaciones aleatorias de 0 a 180 grados. Esto se hizo tres veces para incrementar el número de flores campanuláceas e igualar la cantidad de muestras de las flores rosáceas.

3.2.4. Preprocesamiento de los datos

Las fotografías fueron preprocesadas para el entrenamiento e inferencia de cada uno de los modelos. Este preprocesamiento incluyó el redimensionamiento de las imágenes a un tamaño de 640 x 640, ya que los modelos fueron preentrenados con fotos de dichas dimensiones.

Por otro lado, como se utilizó la biblioteca de Pytorch para el desarrollo del modelo *Faster R-CNN* y el clasificador de flores, fue necesario transformar las imágenes a un formato de tensor. Esto es un requerimiento de implementación de dicha biblioteca para manipular la imágenes y que puedan ser consumidas por los modelos.

3.3. Módulo de estimación de longitud

El objetivo de este módulo es estimar la longitud de las varetas de duraznero presentes en la imagen. Para esto, se utilizó el enfoque de encontrar un objeto de referencia al que se le conocen sus dimensiones en centímetros y, en base a estas dimensiones, se calculan las del elemento objetivo que en este caso son las varetas.

Las fotos provistas por el INTA, como se menciona en la sección 3.2.1, contienen una regla de 30 centímetros de largo y 5 de ancho. Esta regla se utilizó como objeto de referencia para la estimación de longitud de las varetas. Para su detección, se tuvo que aplicar *transfer learning* y *fine tunning* a un modelo de detección. El modelo base utilizado fue *YOLOv8n* de Ultralytics que fue preentrenado con el conjunto de datos de COCO.

Una vez detectada la regla, se toman las dimensiones del *bounding box* y se calcula su perímetro en píxeles. Luego, sabiendo que el perímetro en centímetros es 70, se hace una división del perímetro en píxeles entre el perímetro en centímetros. Esta operación termina dando como resultado, la razón matemática que representa la cantidad de píxeles equivalentes a 1 centímetro. Con esto, se obtuvo la conversión para poder aplicarla a los demás objetos en la imagen.

Posteriormente, se realiza la detección de las varetas, donde se utilizó *YOLOv8n* de Ultralytics preentrenado. En este caso, al igual que para el detector de regla, el *bounding box* proporcionado durante el etiquetado, es lo más preciso con el contorno real del objeto. Esta condición fue necesaria, debido a que las dimensiones de este delimitador son fundamentales para la estimación de la longitud de la vareta.

Cuando se detectan las varetas, se hace la conversión de píxeles a centímetros tomando los anchos y altos de los *bounding boxes* y dividiéndolos entre la razón matemática obtenida anteriormente. El resultado encontrado es la estimación en centímetros del alto y ancho de los *bounding boxes*. Con este dato, se toman en cuenta dos cosas. Si el ancho es más grande que el alto en los *bounding boxes*, entonces, la imagen tiene una orientación horizontal. Por lo tanto, el ancho en este caso es la estimación de longitud tanto para la regla como para las varetas. En caso contrario, la imagen se encuentra en orientación vertical y el alto es efectivamente la estimación de longitud de los objetos.

Por otro lado, cabe destacar que las detecciones de las varetas se organizan de menor a mayor recorriendo la imagen en el eje X (izquierda a derecha), si la imagen es vertical. En caso contrario, se hace de la misma forma, pero sobre el eje Y. Con esta técnica es posible identificar las varetas que se detectan y su respectiva estimación de longitud.

Por último, se imprime por pantalla la imagen con las detecciones y las mediciones, además se muestra una tabla con los resultados obtenidos en este módulo. Esta tabla posteriormente se une con el resultado del módulo de detección y procesamiento. En la figura 3.9 se observa un ejemplo del resultado obtenido por pantalla.

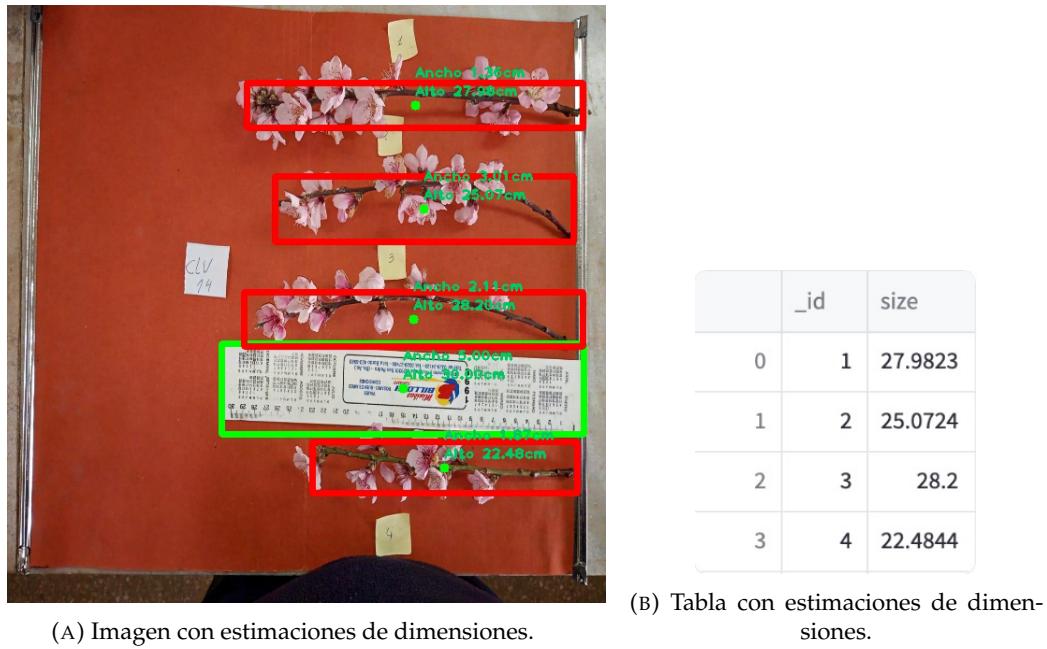


FIGURA 3.9. Resultados del módulo de estimación de longitud.

3.4. Módulo de detección y procesamiento

El propósito de este módulo es detectar los estados fenológicos de las flores de duraznero, hacer un conteo total de la cantidad de flores presentes y por vareta, y por último clasificarlas en campanuláceas o rosáceas.

3.4.1. Detección de estados fenológicos

Para la detección de los estados fenológicos se utilizó el modelo de detección *YOLOv8n* de Ultralytics, preentrenado con el conjunto de datos de COCO. Se le aplicó *fine tuning* para ajustarlo a este caso de uso y como parte de la búsqueda del mejor modelo para realizar dicha detección, se probó contra *Faster R-CNN* que es un detector de dos etapas. Al final de las pruebas se seleccionó el modelo con mejores métricas de *mAP*, que en este caso fue *YOLOv8n*. En el siguiente capítulo se detallarán dichas pruebas.

Los resultados de esta detección no solo incluyen la localización e identificación de cada estado fenológico de cada flor, sino también la detección de la vareta. El motivo para detectar nuevamente la vareta va relacionado a la toma de regiones de interés que serán de utilidad más adelante en este mismo módulo.

En la figura 3.10 se pueden observar los resultados que se obtienen del detector de estados fenológicos de la flor de duraznero.

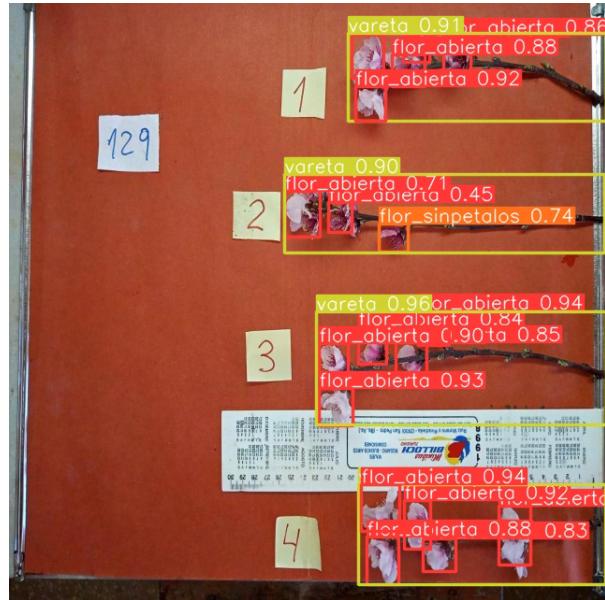


FIGURA 3.10. Imagen de salida del detector de estados fenológicos.

3.4.2. Conteo de flores

Para realizar el conteo total de flores por imagen, primero se posprocesan las detecciones obtenidas con el detector de estados fenológicos. Luego, se buscan los centroides de cada *bounding box* y se filtran las detecciones pertenecientes a algunas de las siguientes clases: flor abierta, flor cerrada, flor sin petalos e incierto. Posteriormente, a través de un script, se crea una lista de flores detectadas y se utilizan los centroides tomadas anteriormente para evitar la toma de detecciones que tuvieran una misma posición. De esta forma, no se toman en cuenta las detecciones duplicadas.

Para realizar el conteo de flores por vareta, se requirió de otro posprocesamiento de los resultados obtenidos del detector de estados fenológicos. En este caso, se toman en cuenta las detecciones pertenecientes a la clase vareta y, al igual que en el módulo que estima la su longitud, se busca el ancho y el alto de los *bounding boxes* de esta clase para, posteriormente, identificar si la imagen posee una orientación horizontal o en vertical. El dato de la orientación de la imagen es importante para conocer como se tiene que recorrer la foto, de forma que se mantenga el mismo identificador de vareta que se utiliza en el módulo de estimación de longitudes.

Una vez que se conoce la orientación de la imagen, se ordenan las detecciones de las varetas y se empieza a recorrer la foto en el eje X si es vertical o en el eje Y si es horizontal. En cada iteración se crea un identificador para cada vareta encontrada. Durante estas iteraciones, para aislar cada vareta de forma de poder contar sus flores, se tuvo que utilizar el enfoque de buscar y crear regiones de interés.

La región de interés se genera a partir de las coordenadas del *bounding box* perteneciente a la clase vareta. Una vez obtenidas dichas coordenadas, se oscurece la imagen pasando cada píxel a cero y se encienden los píxeles dentro de la región de interés. Posteriormente, se realiza una operación *bitwise and* entre la imagen

con la región de interés y la original, dando como resultado la imagen oscurecida con solo una vareta iluminada. Este procedimiento lo realiza el script por cada vareta encontrada y en el orden mencionado anteriormente.

Adicionalmente, por cada región de interés, se procede a filtrar las centroides pertenecientes a dicha región. De esta forma, se genera una lista con las flores encontradas que representa la cantidad de flores por vareta. En la figura 3.11 se puede observar un ejemplo del resultado de esta operación.

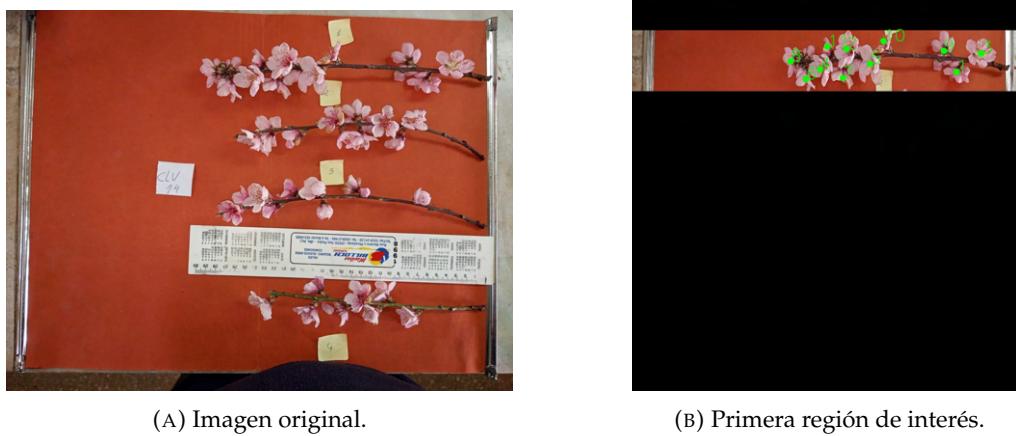


FIGURA 3.11. Ejemplo de conteo de flores por vareta con regiones de interés.

Cabe destacar que para que este procedimiento sea efectivo, es necesario que cada vareta tenga una distancia apropiada entre sí. Bajo esta condición, se evita tomar las flores pertenecientes a una vareta vecina.

3.4.3. Clasificador de flores

El clasificador de flores de duraznero es la última fase que recorre el sistema antes de finalizar el proceso y generar la tabla final de resultados. El desarrollo de este componente requirió diseñar una red neuronal convolucional para hacer distinción entre flores del tipo rosáceas o campanuláceas. En este caso, no se realizó *transfer learning* dado que la clasificación no era muy compleja y se tenía una gran cantidad de muestras para el entrenamiento. En la tabla 3.3 se puede observar la arquitectura de esta red neuronal convolucional.

TABLA 3.3. Arquitectura de CNN para clasificación de flores de duraznero.

Capaz	Dimensiones de salida	Parámetros
Conv2d-1.	[-1, 16, 25, 25]	448
MaxPool2d-2.	[-1, 16, 12, 12]	0
Conv2d-3.	[-1, 32, 12, 12]	4640
MaxPool2d-4.	[-1, 32, 6, 6]	0
Conv2d-5.	[-1, 64, 6, 6]	18 496
MaxPool2d-6.	[-1, 64, 3, 3]	0
Linear-7.	[-1, 128]	73 856
Dropout-8.	[-1, 128]	0
Linear-9.	[-1, 2]	258

Esta red convolucional cuenta con un total de 97 698 parámetros entrenables. Por otro lado, se aplicaron métodos como regulación L2 [28] y *DropOut* [29] para disminuir el sobreajuste. Las pruebas del rendimiento de este modelo y el sobre ajuste se explicarán con más detalle en el capítulo siguiente.

Este modelo se integra y se ejecuta luego de encontrar y delimitar la región de interés en cada vareta. Al momento de tomar la región de interés y contar la cantidad de flores en dicha región, se filtra por aquellas detecciones que representan a las clases flor abierta y flor cerrada, para posteriormente recortar dicha detección utilizando sus coordenadas provistas por el *bounding box*. Una vez recortada la imagen de la flor, se redimensiona a una resolución de 25 x 25 y se envía al clasificador. Finalmente, a través de su última capa totalmente conectada, el clasificador produce, una probabilidad para cada clase y luego se selecciona la clase con la probabilidad más alta. Este proceso se repite por cada flor abierta y cerrada encontrada en la vareta.

Al final, cuando se clasificaron todas las flores abiertas y cerradas de una misma vareta, se guarda el resultado en un arreglo de datos y se toma la categoría con mayor aparición en dicho arreglo. Es decir, este arreglo permite que si el clasificador se equivoca categorizando una flor, se pueda enmendar dicho error con un proceso de votación. Este paso es importante porque una misma vareta no puede tener más de una clase.

Por otro lado, el proceso iterativo de recorrer la imagen y clasificar cada vareta con un tipo de flor, permitirá que en el futuro se pueda tener una imagen que contenga varetas con distintas categorías.

3.5. Integración de módulos y generación de resultados

La integración de los módulos se realizó a través del lenguaje de programación Python, donde ambos componentes se ejecutan desde el script principal y de forma secuencial. De esta forma, como se mostró en la figura 3.1, el primer módulo en recibir la imagen es el de estimación de longitud y posteriormente se ejecuta el de detección y procesamiento.

En cada módulo se genera un diccionario de Python, que contiene el resultado de dicha operación, ambos resultados incluyen una variable en común. Esta variable es la identificación de cada vareta que se produce durante el recorrido que realiza cada algoritmo sobre la imagen. Por este motivo, es importante que ambos algoritmos recorran la imagen de la misma forma para que esta variable sea consistente en ambos resultados.

En un paso posterior, los resultados de cada algoritmo se transforman a un formato de *dataframe* de Pandas para luego concatenarlos y presentar al usuario final una única tabla de resultados.

En la figura 3.12 y la figura 3.13 se observa un ejemplo de la tabla resultante de cada algoritmo presentada por pantalla.

	_id	size
0	4	31.6709
1	3	22.6878
2	2	18.5798
3	1	16.5254

FIGURA 3.12. Ejemplo de tabla resultante del módulo de estimación de longitud.

_id	coordenadas	num_flores	flores_abiertas	flores_cerradas	flores_sinpétalos	incerto	tipo_de_flor	num_total_flores	num_total_varetas
0	4	33.449310302734375	195.6981964111328	145.24270629882812	596.865173	7	7	0	0
1	3	236.88287353515625	320.3204345703125	338.58746337890625	593.975463	10	7	2	1
2	2	351.96234130859375	351.296630859375	446.76751708984375	591.9935302	7	2	0	5
3	1	512.6819458007812	364.4129638671875	609.5979614257812	586.71472167	7	4	0	3

FIGURA 3.13. Ejemplo de tabla resultante del módulo de clasificación y procesamiento.

Cabe destacar que se generaron dos botones para obtener la tabla de resultados en dos formatos distintos. El primero genera el resultado en formato CSV y el segundo produce el resultado en formato XLSX para su correcta visualización en Excel.

En la figura 3.14 se muestra un ejemplo de ambas tablas unificadas que representan el resultado final del sistema.

_id	size	coordenadas	num_flores	flores_abiertas	flores_cerradas	flores_sinpétalos	incerto	tipo_de_flor	num_total_flores	num_total_varetas
4	31.67096485309424	(33.44931, 195.6982, 145.2427, 596.8652)	7	7	0	0	0	Flor_Rosacea		
3	22.6877705020939576	(236.88287, 320.32043, 338.58746, 593.97546)	10	7	2	1	0	Flor_Rosacea		
2	18.579820583164344	(351.96234, 351.29663, 446.76751, 591.99353)	7	2	0	5	0	Flor_Rosacea		
1	16.525442349088053	(512.68194, 364.41296, 609.59796, 586.71472)	7	4	0	3	0	Flor_Rosacea	33.0	4.0

FIGURA 3.14. Ejemplo de tabla unificada.

3.6. Interfaz de usuario

La interfaz gráfica del programa fue realizada con Streamlit [30], que es una herramienta que permite compartir aplicaciones *web*, utilizando Python. Adicionalmente, Esta herramienta no requiere ningún conocimiento previo en desarrollo de *front end*.

Para el desarrollo del *front end* con Streamlit y para ejecutar el proyecto en un entorno local, fue necesario crear una estructura de directorios que contuviesen los modelos de IA, los *scripts* de Python para ejecutar cada módulo, los *logs* generados por el sistema y por último, para tener un conocimiento de los experimentos realizados, un directorio para guardar las *notebooks* utilizadas. Para su ejecución, se creó un entorno virtual con Python, que contiene todas las bibliotecas y dependencias necesarias.

Los *scripts* de Python son cuatro en total, donde dos ellos llamados densidad vareta.py y vareta size.py representan los módulos de detección y procesamiento, y estimación de longitud respectivamente. El *script* de flower classifier.py contiene la red neuronal convolucional desarrollada para el clasificador de flores. Por último, se tiene el *script* principal que ejecuta la interfaz de usuario con Streamlit y orquesta los módulos del sistema.

La interfaz de usuario cuenta con una página de presentación, donde se observa el título del trabajo, las tecnologías utilizadas y su propósito. En la figura 3.15 se muestra un ejemplo de esta pantalla de inicio.

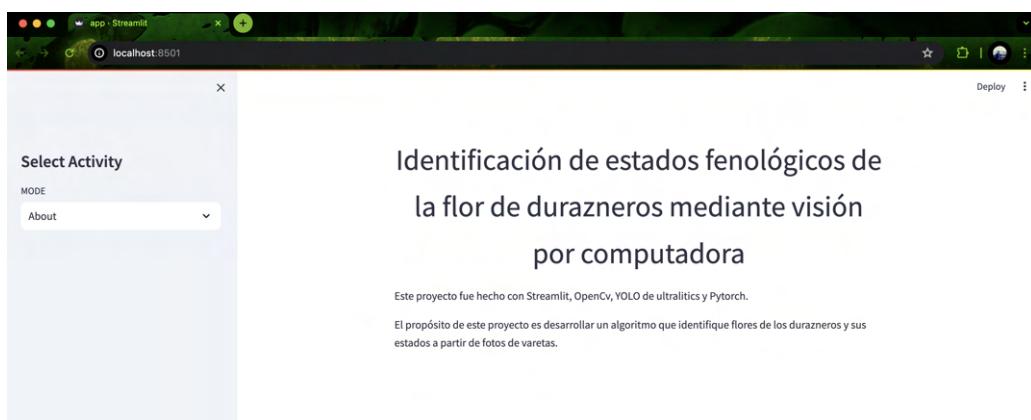


FIGURA 3.15. Pantalla de inicio del programa.

Como el programa se ejecuta en un entorno local, Streamlit, utiliza el *localhost* y el puerto 8501 por defecto para desplegar el sitio *web*.

En el panel izquierdo se puede observar un desplegable que permite seleccionar el tipo de actividad. Para este desarrollo solo existe la detección de objetos. Al seleccionar detección de objetos, se muestra la siguiente página, donde se pedirá insertar una foto de 200 MB de tamaño máximo, en los formatos JPG, PNG o JPEG. La imagen se puede arrastrar directamente hasta el rectángulo de carga o se puede presionar directamente el botón de *Browse files*. En la figura 3.16 se presenta un ejemplo de esta segunda página destinada para subir imágenes.

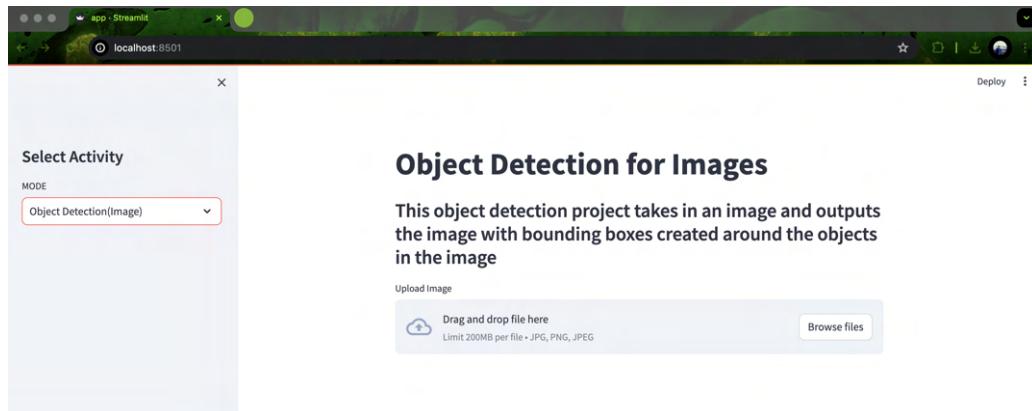


FIGURA 3.16. Página para cargar imágenes.

Al subir una imagen, se muestra la fotografía subida y se habilita una barra que modifica el parámetro de nivel de confianza (que va de un intervalo de 0 a 1) del algoritmo que detecta las varetas para el módulo de estimación de longitud y luego se habilita otra barra con el mismo propósito pero para el segundo módulo. El objetivo de estas barras es darle al INTA la flexibilidad de hacer que el algoritmo sea más estricto con las detecciones o más flexible. Por otro lado, cada barra inicia en un valor por defecto, en el caso del módulo de estimación de longitud, este valor es de 0,6 y el módulo de detección y procesamiento es de 0,4. Se seleccionaron estos valores por defecto porque son los valores donde ambos módulos en general tienen un buen rendimiento.

En las figuras 3.17 y 3.18 se muestran dichas barras con una foto de vareta de prueba.

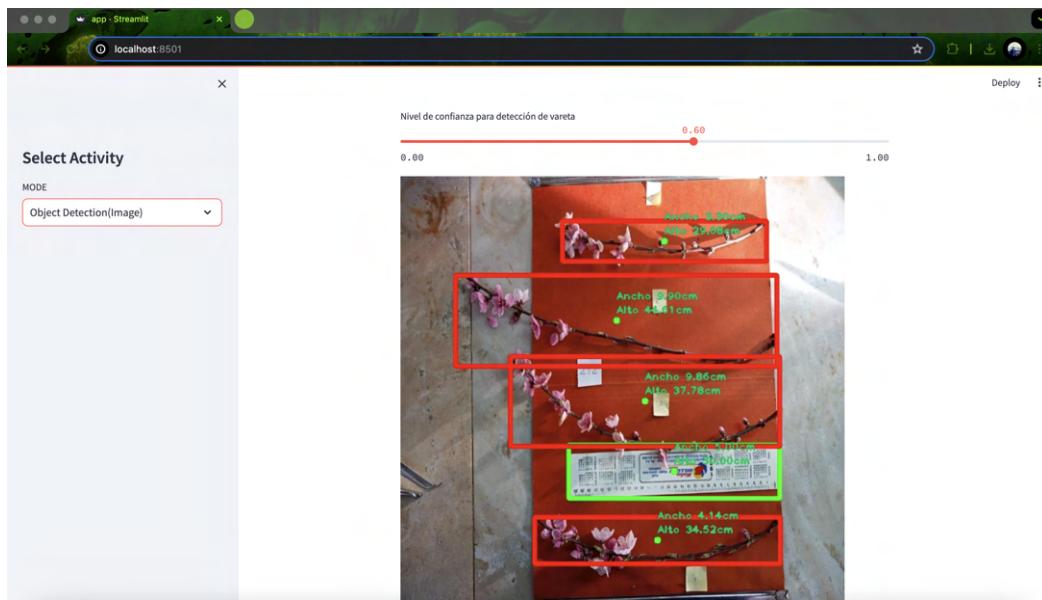


FIGURA 3.17. Barra de confianza para el módulo de estimación de longitud.

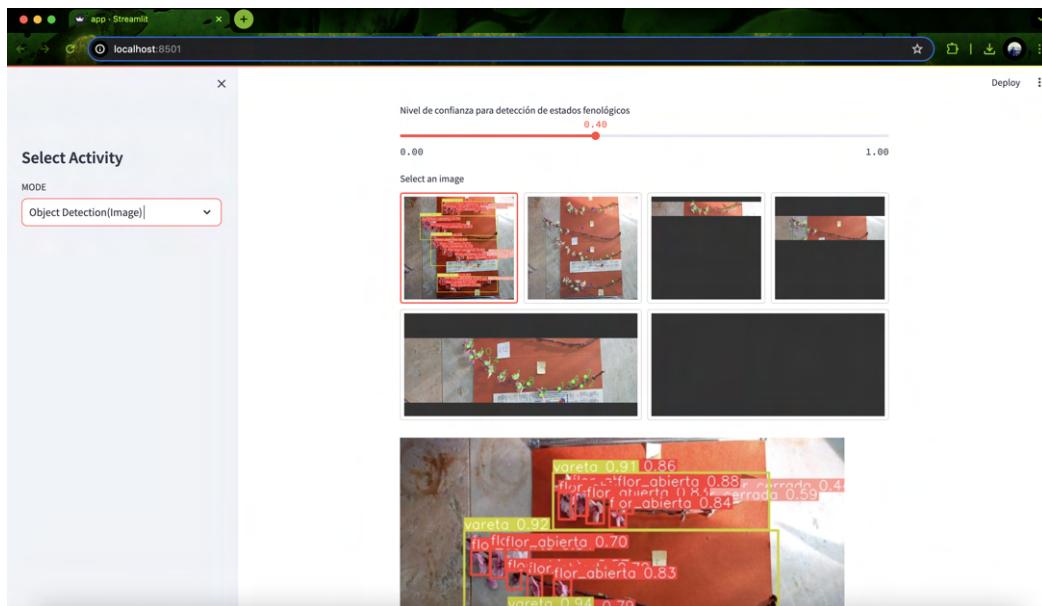


FIGURA 3.18. Barra de confianza para el módulo de detección y procesamiento.

Por último, al final de toda la ejecución como se mencionó anteriormente, se presentan dos botones que permiten descargar los resultados en los formatos CSV y Excel respectivamente. En la figura 3.19 se pueden observar dichos botones.

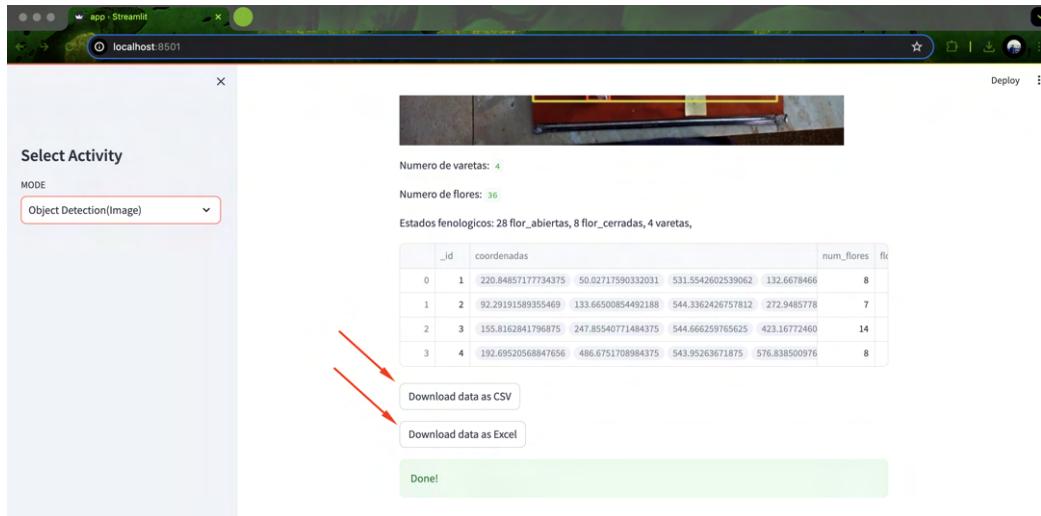


FIGURA 3.19. Botones de descarga de resultados.

Capítulo 4

Ensayos y resultados

En este capítulo se describen los ensayos realizados y se presentan y comparan los resultados obtenidos.

4.1. Banco de pruebas

Para el desarrollo del presente proyecto se utilizaron 3 bancos de pruebas, con las siguientes especificaciones técnicas:

- Banco de pruebas Laptop MacBook Pro - *hardware* disponible:
 - OS: MacOS Sonoma 14.
 - CPU: Apple M1 Pro.
 - RAM: 32 GB.
 - GPU: No contiene.
- Banco de pruebas Laptop ASUS TUF GAMING F15 - *hardware* disponible:
 - OS: Windows 11.
 - CPU: Intel(R) Core(TM) i5-11260H.
 - RAM: 16 GB.
 - GPU: NVIDIA GeForce RTX 3050.
- Banco de pruebas Google Colab Pro - *hardware* disponible:
 - OS: desconocido.
 - CPU: desconocido.
 - RAM: 12,5 GB.
 - GPU: T4.

4.2. Desempeño de modelos

Para medir el desempeño de los modelos de IA utilizados durante la ejecución de este trabajo, se decidió emplear la métrica mAP. Esta métrica evalúa tanto la clasificación del objeto como la ubicación del *bounding box* a través del *Average Precision* (AP) que se representa en la ecuación 4.1.

$$\text{AP} = \frac{1}{n} \sum_{k=1}^n \text{Precisión en } k \times \text{Rel}_k \quad (4.1)$$

Donde

- n es el número total de elementos recuperados.
- Precisión en k es la precisión en el k -ésimo punto de recuperación.
- Rel_k es un indicador binario que denota si el elemento en el k -ésimo punto de recuperación es relevante ($\text{Rel}_k = 1$) o no relevante ($\text{Rel}_k = 0$).

Luego, al promediar los valores de AP entre todas las clases, se obtiene respectivamente el mAP y la ecuación queda como se muestra en 4.2.

$$\text{mAP} = \frac{1}{C} \sum_{c=1}^C \text{AP}_c \quad (4.2)$$

Donde

- C es el número total de clases o consultas.
- AP es el *Average Precision* calculado para la clase o consulta.

4.2.1. Desempeño del detector de regla

Como se mencionó en el capítulo anterior, el detector de la regla tiene como modelo base un *YOLOv8n* y cumple una función fundamental para la estimación de la longitud de la vareta. Por otro lado, el entrenamiento de este modelo se realizó con los siguientes hiperparámetros:

- Épocas: 100.
- Tamaño de imagen de entrada: 640×640 .
- *Batch*: 16.
- *Learning rate*: 0,01.

Los resultados obtenidos para la detección de este elemento contra el conjunto de datos de prueba se observan en la tabla 4.1.

TABLA 4.1. Métricas de detección para el detector de regla.

Clase	Imágenes	Detecciones	Precision	Recall	mAP50	mAP50-95
Regla	13	13	0,996	1,0	0,995	0,995

Los resultados fueron muy buenos en general. El modelo puede detectar la regla sin dificultad en el 99 % de los casos. Por este motivo, se tomó directamente este modelo para desempeñar la tarea y no se requirió de aumentos de datos o alguna otra técnica de *machine learning*.

4.2.2. Desempeño del detector de vareta

El detector de vareta destinado para el módulo de estimación de longitud, tiene como modelo base un *YOLOv8n* entrenado bajo los siguientes hiperparámetros:

- Épocas: 100.
- Tamaño de imagen de entrada: 640 x 640.
- *Batch*: 16.
- *Learning rate*: 0,01.

Para este modelo se hicieron dos pruebas, una sin aumento de datos y otra con aumento. El aumento de datos aplicado corresponde a lo descripto en la sección 3.2.3.

Los resultados obtenidos contra el conjunto de datos de pruebas para el modelo sin aumento de datos se muestran en la tabla 4.2.

TABLA 4.2. Métricas de detección para el detector de varetas sin aumento de datos.

Clase	Imágenes	Detecciones	Precision	Recall	mAP50	mAP50-95
Vareta	10	40	1,0	0,923	0,983	0,574

Como se puede observar, el modelo sin aumento de datos consigue detectar un total de 40 varetas en 10 imágenes, donde todas las detecciones fueron correctas. Por otro lado, el modelo logra una precisión promedio de detección del 98,3 % cuando se requiere un nivel de confianza igual al 50 %, pero el mAP 50-95, indica que en los casos donde el nivel de confianza esta entre el 50 % y 95 %, la precisión promedio, es de un 57,4 %. Por este motivo, se decidió aplicar aumento de datos.

Los resultados con aumento de datos se pueden observar en la tabla 4.3.

TABLA 4.3. Métricas de detección para el detector de varetas con aumento de datos.

Clase	Imágenes	Detecciones	Precision	Recall	mAP50	mAP50-95
Vareta	10	40	0,974	0,975	0,976	0,702

Luego de aplicar aumento de datos, se observa que el modelo mejoró la cantidad de veces donde detecta de forma precisa a la clase vareta en un 5,2 % con respecto al modelo sin aumento de datos. Por otro lado, también se incrementó la precisión promedio al 70,2 % cuando se tiene un umbral de confianza que va entre 50 % y 95 %.

Con esto se destaca que el aumento de datos mejoró el rendimiento del modelo de forma considerable.

4.2.3. Desempeño del detector de estados fenológicos

La detección de estados fenológicos, como se mencionó en el capítulo anterior, fue probada con dos modelos de IA que se ajustan para este caso de uso. Por un lado, se probó con un detector de dos etapas como es *Faster R-CNN* y por otro

lado con un detector de una etapa como YOLOv8. Ambos casos, se probaron con y sin el uso de aumento de datos. Los resultados fueron los siguientes:

Faster R-CNN sin aumento de datos

El entrenamiento del modelo se hizo bajo los siguientes hiperparámetros:

- Épocas: 100.
- Tamaño de imagen de entrada: 640 x 640.
- *Batch*: 4.
- *Learning rate*: 0,001.
- *Momentum*: 0,9.
- *Weight decay*: 0,0005.
- *Optimizer*: *Stochastic gradient descent*.

Los resultados de detección contra el conjunto de prueba se muestra en la tabla 4.4.

TABLA 4.4. Métricas de detección para *Faster R-CNN* sin aumento de datos.

Clase	Imágenes	mAP	mAP50	mAP>50
Todas	14	0,3492	0,5970	0,3605
Flor abierta	14	0,4313	-	-
Flor cerrada	14	0,1525	-	-
Flor sin pétalos	14	0,3735	-	-
Incierto	14	0,0942	-	-
Vareta	14	0,6947	-	-

La precisión promedio de detección entre todas las clases es de un 34,9 %, lo que significa que el modelo tiene una precisión del 34,9 % al detectar las diferentes clases en el conjunto de datos. Además, el modelo logra una precisión de detección del 59,7 % cuando se considera un umbral de confianza igual al 50 % y para detecciones con un nivel de confianza mayor al 50 % se tiene una precisión del 36,1 %.

Por otro lado, al evaluar la precisión promedio de detección por clase, se observa que la categoría vareta es la clase a la que el modelo detecta con mejor precisión promedio con 69,5 %, seguida por flor abierta con 43,13 % y por flor sin pétalos con 37,35 %. Además, se muestra que el modelo detecta con dificultad las categorías flor cerrada con un mAP de 12,25 % e incierto con 9,42 %.

Este bajo porcentaje de mAP para la clase *flor cerrada* con respecto a las otras categorías presentes, se puede deber al desbalance de datos observado en la sección 3.2.2 y al rendimiento del modelo al intentar detectar objetos pequeños en la imagen. Adicionalmente, para la categoría *incierto* no se cuenta con una cantidad de muestras significativas y consistente a través del conjunto de datos, lo que explica la baja precisión promedio obtenida para esta clase.

Para mejorar estos resultados se implementó el aumento de datos mencionado en la sección 3.2.3.

Faster R-CNN con aumento de datos

El rendimiento del modelo basado en la métrica mAP, se puede observar en la tabla 4.5.

TABLA 4.5. Métricas de detección para *Faster R-CNN* con aumento de datos.

Clase	Imágenes	mAP	mAP50	mAP>50
Todas	14	0,3733	0,6328	0,3793
Flor abierta	14	0,5143	-	-
Flor cerrada	14	0,2049	-	-
Flor sin pétalos	14	0,3290	-	-
Incierto	14	0,0887	-	-
Vareta	14	0,7294	-	-

Como se puede observar, se logra mejorar la precisión promedio de detección para las clases flor abierta, flor cerrada y vareta. Con esto, la métrica mAP medida entre todas las clases mejora a 37,33 %, adicionalmente también aumenta a 63,26 % cuando se fija un umbral de confianza de detección igual al 50 % y a un 37,93 % cuando el nivel de confianza en la detección es superior al 50 %.

Se puede concluir que el modelo *Faster R-CNN* con y sin aumento de datos es bueno para detectar la vareta y las flores en estado abierto. Este comportamiento se puede deber al tamaño de estos objetos en la imagen en comparación con el resto de las categorías que se busca en la detección. Además, la clase flor abierta es la clase más predominante en todas las imágenes según el estudio de los datos realizado en la sección 3.2.2, lo que permite que el modelo pueda aprender y reconocer con mayor precisión sus características debido a la abundancia de ejemplos representativos durante el entrenamiento.

YOLOv8n sin aumento de datos

Se puede observar en la tabla 4.6 el resultado obtenido con *YOLOv8n* sin aumento de datos, con 14 imágenes del conjunto de pruebas y bajo los siguientes parámetros de entrenamiento:

- Épocas: 100.
- Tamaño de imagen de entrada: 640 x 640.
- *Batch*: 16.
- *Learning rate*: 0,01.

TABLA 4.6. Métricas de detección para YOLOv8n sin aumento de datos.

Clase	Imágenes	mAP50	mAP>50
Todas	14	0,645	0,418
Flor abierta	14	0,905	0,611
Flor cerrada	14	0,614	0,251
Flor sin pétalos	14	0,588	0,363
Incierto	14	0,122	0,065
Vareta	14	0,994	0,802

Se puede observar que la precisión promedio de todas las clases obtenida por YOLOv8n es de 64,5 % para el mAP 50. Lo que supera al mAP 50 obtenido con el modelo *Faster R-CNN* con y sin aumento de datos.

Las clases que este modelo detecta con más precisión, son: vareta, flor abierta y flor cerrada. La diferencia y mejora contra *Faster R-CNN* se puede observar también al comparar la métrica mAP obtenida para todas las clases donde este modelo se desempeñó mejor. Como ejemplo se tiene la clase vareta, donde se obtuvo un 69,5 % con *Faster R-CNN*. Esta misma clase que también fue la mejor para el modelo YOLOv8n, tuvo un 99,4 % que representa una mejora del 29,9 %.

Por otro lado, ambos modelos logran detectar con alta precisión la clase vareta y esto se puede deber al tamaño de este objeto en la imagen. Además, es seguida por la clase flor abierta que, como se mencionó anteriormente, contiene una cantidad de muestras representativas que ayudan al modelo a reconocer los patrones de esta clase. Por último, se destaca la categoría incierto, que tanto este modelo como *Faster R-CNN* no logran reconocer durante la detección en su totalidad.

La matriz de confusión normalizada mostrada en la figura 4.1, indica que muchas de las veces que se tiene la clase incierto, el modelo la predice como flor abierta. Esto demuestra que el modelo no logró aprender correctamente los patrones que identifican dicha clase durante el entrenamiento.

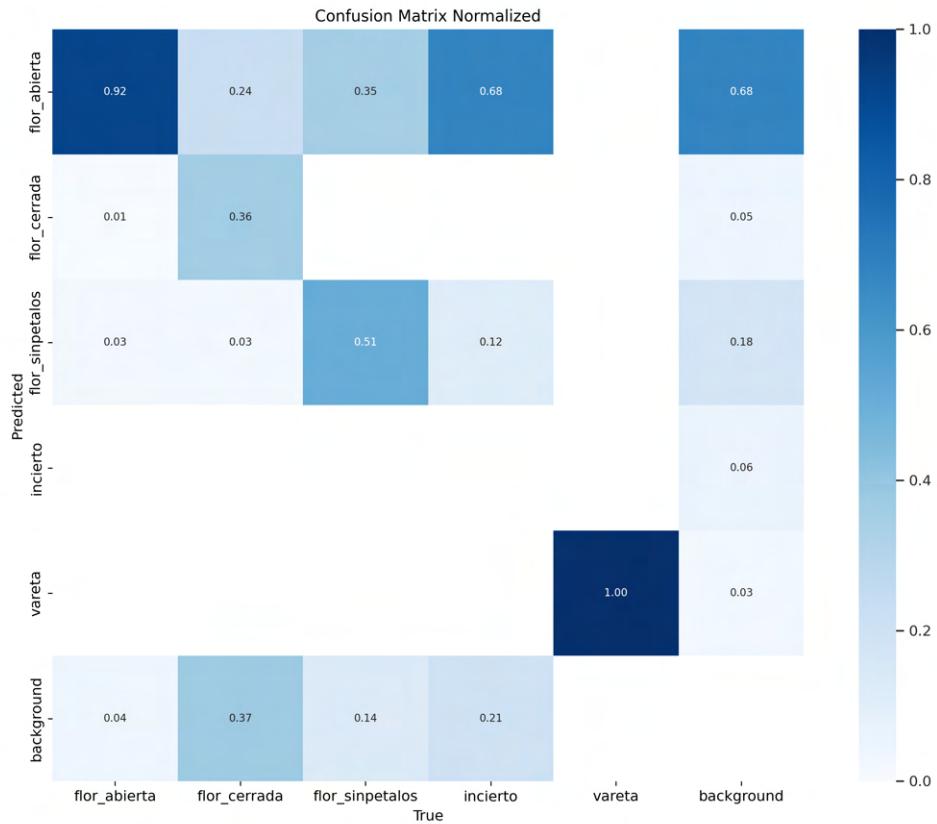


FIGURA 4.1. Matriz de confusión normalizada de *YOLOv8n* para el conjunto de prueba.

YOLOv8n con aumento de datos

Para mejorar los resultados de *YOLOv8n* se implementó el aumento de datos descrito en la sección 3.2.3. Los resultados obtenidos contra el conjunto de imágenes de prueba, se observa en la tabla 4.7.

TABLA 4.7. Métricas de detección para *YOLOv8n* con aumento de datos.

Clase	Imágenes	mAP50	mAP>50
Todas	14	0,655	0,423
Flor abierta	14	0,893	0,611
Flor cerrada	14	0,673	0,266
Flor sin pétalos	14	0,603	0,373
Incierto	14	0,111	0,065
Vareta	14	0,995	0,802

La precisión promedio para un umbral de confianza igual al 50 % mejora para la clase flor cerrada en un 5,9 % y para la clase flor sin pétalos en un 1,5 %. Mientras que, se tiene una perdida para la clase flor abierta del 1,2 %. De esta forma, se observa un aumento del 1 % para el mAP 50 de todas las clases.

Se observa que el entrenamiento del modelo con aumento de datos ayudó a mejorar el reconocimiento de las clases más difíciles de detectar como son flor cerrada y flor sin pétalos. Además, mantiene el buen rendimiento para las clases flor abierta y vareta.

Al comparar todos los experimentos realizados para la detección de los estados fenológicos de las flores de duraznero, el modelo con mejor rendimiento basado en la métrica mAP para dicha tarea es *YOLOv8n* con aumento de datos. Por este motivo, se seleccionó este modelo para desempeñar dicha tarea. Adicionalmente, este modelo es capaz de mitigar el desbalance de clases por su mecanismo interno que hace uso del *Focal Loss* [23] para asignar mayor peso a las clases menos frecuentes en forma automática.

Por último, se aplicó *Eigen-CAM* [31] para crear mapas de calor que permitan explicar los resultados obtenidos por el modelo. Este mapa de calor se implementó en la capa convolucional *C2f*. En la figura 4.2 se observa un ejemplo de los mapas de calor generados por *Eigen-CAM* en una foto perteneciente al conjunto de pruebas utilizando *YOLOv8n* con aumento de datos.

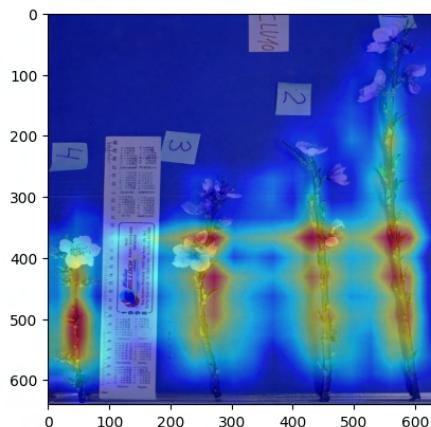


FIGURA 4.2. Mapa de calor con *Eigen-CAM* en la capa *C2f* del modelo *YOLOv8n*.

Se observa que los mapas de activación están enfocados en las varetas para la capa convolucional seleccionada, lo que sugiere que el modelo pone mayor atención en este objeto al momento de realizar la detección.

4.3. Desempeño de módulos

Como se mencionó en secciones anteriores, el sistema cuenta con dos módulos, y para estimar su desempeño se utiliza un método visual. Este método implica la introducción de una imagen o fotografía en el sistema, de manera que esta transite a través de cada módulo de manera secuencial. Posteriormente, se realiza una observación de la salida generada por cada módulo de forma individual y se toman conclusiones de su funcionamiento. Esta aproximación no solo facilita la medición del rendimiento del sistema en su totalidad, sino que también permite un examen detallado de las contribuciones y funcionalidades específicas de cada uno de sus componentes modulares.

4.3.1. Desempeño del módulo de estimación de longitud

El funcionamiento y precisión del módulo de estimación de longitud está relacionado a la calidad de las imágenes que recibe a la entrada. Estas imágenes tienen que contar con la regla de 30 centímetros como objeto de referencia y con condiciones similares a las que se presentaron en el entrenamiento del modelo (ejemplo: fondo, separación entre varetas, etc).

Por otro lado, en esta prueba de desempeño, se toma en cuenta la medición del objeto de interés y visualmente se compara la medición de la vareta más cercana a la regla, donde se podrá observar si su longitud visual es parecida a la estimada por el algoritmo. Además, se destaca que la medida se presenta en centímetros.

En la figura 4.3, se observa la prueba realizada para conocer el desempeño de este módulo.



FIGURA 4.3. Desempeño del módulo de estimación de longitud.

Como se puede observar, la vareta con la etiqueta número cuatro mide 26,27 centímetros según el algoritmo. Sin embargo, visualmente aparece medir 27 centímetros aproximadamente, lo que sugiere un error de medición de 0,73 centímetros. Adicionalmente, si se observa la vareta con la etiqueta número tres, su longitud según el algoritmo es de 29,60 centímetros, pero visualmente se aproxima a 30 centímetros, por lo que se presume un error de 0,4 centímetros.

Con este resultado se puede concluir que el módulo approxima la longitud de las varetas a valores muy cercanos a los que se obtienen visualmente. Sin embargo, al no poder medir con exactitud la longitud real de ambas varetas no se puede obtener con precisión el rendimiento de este módulo. Además, cabe destacar que esta medida se obtiene de forma lineal y no se cuentan las curvaturas de las varetas, lo que implica que esta medida no es exacta.

4.3.2. Desempeño del módulo de detección y procesamiento

Este módulo, como se mencionó en la sección anterior, realiza múltiples operaciones donde detecta los estados fenológicos, hace un conteo de la cantidad de flores en la imagen y por vareta, cuenta la cantidad de varetas y determina la cantidad de flores que se tienen en un estado o en otro.

En la figura 4.4, se muestra una foto de varetas con los centroides pertenecientes a las detecciones realizadas. De este forma, es más simple evaluar y realizar el conteo de flores de forma visual, luego compararlo con los resultados obtenidos por el algoritmo que se detallan en la tabla 4.8.



FIGURA 4.4. Desempeño del módulo de detección y procesamiento con el uso de centroides con flores campanuláceas.

TABLA 4.8. Tabla de resultados del módulo de detección y procesamiento para la imagen de muestra.

Id	Nº flores	Abiertas	Cerradas	Sin pétalos	Tipo
1	7	6	1	0	Campanulácea
2	15	13	2	0	Campanulácea
3	12	9	2	0	Campanulácea
4	11	6	5	0	Campanulácea
Total	45	-	-	-	-
Nº varetas	4	-	-	-	-

Se toma como muestra la vareta con el identificador número cuatro, donde el algoritmo indica que existen un total de once flores. Luego, al realizar el conteo

visual, se encuentran un total de trece flores. De estas trece flores, las dos flores que no pudo detectar el algoritmo con sus valores por defecto (nivel de confianza igual a 0,4) son flores cerradas. Por otro lado, el algoritmo señala que de las once flores, hay seis abiertas y cinco cerradas. Visualmente se detectan seis flores abiertas (como menciona el algoritmo) y siete flores cerradas. Con esto, se puede concluir y corroborar que la detección y las tareas derivadas que ejecuta este módulo son impactadas por el rendimiento del modelo base detallado en la sección 4.2.3, donde se observa que el modelo tiene una alta precisión para detectar las flores en estado abierto, pero la precisión es menor para las demás clases relacionadas a los otros estados fenológicos.

Además, de los resultados se destaca que para cada vareta el tipo de flor encontrada por el clasificador fue del tipo campanulácea, lo que coincide con lo observable en la imagen de muestra. Igualmente se remarca el buen funcionamiento del módulo en la detección de las cuatro varetas encontradas en la imagen.

Capítulo 5

Conclusiones

En este capítulo se presentan las conclusiones del trabajo en general, los principales aportes obtenidos y los próximos pasos.

5.1. Conclusiones generales

El presente trabajo cumplió con todos los requerimientos planteados en la sección 1.3, con la salvedad del requerimiento opcional de documentación, donde se planteó documentar en un archivo README.md dentro de un repositorio. Este último requerimiento opcional, no se realizó por falta de tiempo.

En cuanto al cronograma no se cumplió fielmente como se planteó durante la planificación, ya que algunas de las tareas se ejecutaron de forma más rápida que otras y hubo otras que duraron más del tiempo planificado por su complejidad y que requerían mayor dedicación. Las tareas que demandaron mayor cantidad de tiempo y dedicación fueron las siguientes:

- Etiquetado de imágenes.
- Redacción de la memoria final del proyecto.

Por otro lado, a pesar de cumplir con todos los requerimientos funcionales, existió una tarea que no se finalizó en su totalidad y otras dos que no se realizaron. Con esto, la tarea que no se logró finalizar por falta de tiempo fue el etiquetado de todas las imágenes del conjunto de datos. Además, no se ejecutaron las actividades asociadas al desarrollo del *endpoint*, ya que no se requirió su uso al ser un despliegue local.

Durante la ejecución del presente trabajo se manifestaron los siguientes riesgos descritos en la planificación:

- Datos insuficientes. La cantidad de imágenes proporcionadas es insuficiente o su calidad no es la adecuada para entrenar el modelo.
- Etiquetar incorrectamente los estados fenológicos de la flor.

Ambos riesgos fueron mitigados siguiendo plan de mitigación diseñado durante la planificación. Donde el primer riesgo requirió el uso de *transfer learning* y aumento de datos, y para el segundo riesgo se requirió la participación del especialista del INTA para identificar correctamente los estados fenológicos en las fotos proporcionadas.

Por último, se destacan los logros del presente trabajo:

- Se desarrolló un algoritmo capaz de identificar los estados fenológicos de la flor de duraznero y extraer información de su vareta (cantidad de flores, tipo de flor y longitud en centímetros) a partir de imágenes.
- Se diseñó el sistema para funcionar en una computadora local sin la necesidad de GPU y con la posibilidad de funcionar en distintos sistemas operativos.
- Se realizó una interfaz de usuario para su uso y acceso.
- El sistema acepta distintos formatos de imágenes de entrada entre los cuales se pueden mencionar JPG, JPEG, PNG.
- Los resultados se entregan en formato CSV y XLSX para Excel.

5.2. Próximos pasos

La realización del presente trabajo da lugar a múltiples mejoras que se pueden implementar en un futuro próximo. Entre ellas, se sugieren las siguientes:

- Recolectar más fotos de varetas de ambos tipos de flor, especialmente de la clase minoritaria (flor campanulácea).
- Terminar el etiquetado del conjunto de datos y re-entrenar el detector de estados fenológicos para obtener un mejor resultado en las distintas clases.
- Modificar el sistema para que reciba múltiples fotos de entrada.
- Migrar el sistema a un ambiente *Cloud* para agregar escalabilidad y mayor disponibilidad.

Bibliografía

- [1] Maximiliano Martín Aballay; Natalia Cristina Aguirre; Carla Valeria Filippi; Gabriel Hugo Valentini; Gerardo Sánchez. «Fine-tuning the performance of ddRAD-seq in the peach genome». En: *Scientific Reports* (2021).
- [2] DEBORAH PUEBLA.
lujan-y-tunuyan-las-zonas-mas-afectadas-por-las-heladas-tardias.
<https://www.mendozapost.com/sociedad/lujan-y-tunuyan-las-zonas-mas-afectadas-por-las-heladas-tardias/>. Oct. de 2023. (Visitado 12-10-2023).
- [3] T. Dhikhi; Allagada Naga Suhas; Gosula Ramakanth Reddy; Kanadam Chandu Vardhan. «Measuring Size of an Object using Computer Vision». En: *International Journal of Innovative Technology and Exploring Engineering (IJITEE)* (2019).
- [4] Jose Luis Osorio Naranjo. «VIABILIDAD IDENTIFICACIÓN DE ESTADOS FENOLOGICOS EN LA ROSA APLICANDO ALGORITMOS DE RECONOCIMIENTO DE IMAGENES». En: *UNIVERSIDAD EAFIT* (2019).
- [5] J. Dhupia; K. Zhao; R. Li; Y. Cui G. Li; L. Fu; C. Gao; W. Fang; G. Zhao; F. Shi. «Multi-class detection of kiwifruit flower and its distribution identification in orchard based on YOLOv5l and euclidean distance». En: *Comput. Electron. Agric.* (2022).
- [6] G. Li; R. Suo; G. Zhao; C. Gao; L. Fu; F. Shi; J. Dhupia; R. Li; Y. Cui. «Real-time detection of kiwifruit flower and bud simultaneously in orchard using YOLOv4 for robotic pollination». En: *Comput. Electron. Agric.* (2022).
- [7] D. Oñoro-Rubio; R.J. López-Sastre. «Towards perspective-free object counting with deep learning». En: *Computer Vision ECCV 2016, Springer International Publishing* (2016).
- [8] Y. Tian; X. Chu; H. Wang. «Cctrans: simplifying and improving crowd counting with transformer». En: *Computer Vision and Pattern Recognition (cs.CV)* (2021).
- [9] Inc. The MathWorks. ¿Qué son las redes neuronales convolucionales? <https://es.mathworks.com/discovery/convolutional-neural-network.html>. Mar. de 2024. (Visitado 26-03-2024).
- [10] Departamento de Matemática Aplicada. *Redes Neuronales Convoluciones*. https://dcain.etsin.upm.es/~carlos/bookAA/05.7_RRNN_Convoluciones_CIFAR_10_INFORMATIVO.html. Mar. de 2024. (Visitado 26-03-2024).
- [11] IBM. ¿Qué son las redes neuronales convolucionales? <https://www.ibm.com/es-es/topics/convolutional-neural-networks>. Mar. de 2024. (Visitado 26-03-2024).

- [12] Zoumana Keita. *¿Qué son las redes neuronales convolucionales?* <https://www.datacamp.com/es/blog/yolo-object-detection-explained>. Mar. de 2024. (Visitado 27-03-2024).
- [13] Inc. The MathWorks. *Object Detection*. <https://la.mathworks.com/discovery/object-detection.html>. Mar. de 2024. (Visitado 27-03-2024).
- [14] Tony Lindeberg. «Scale Invariant Feature Transform». En: *Scholarpedia* (2012).
- [15] Sergio Hernán Valenzuela Cámara. «Detección y Clasificación de Enfermedades en el Tomate Mediante Deep Learning y Computer Vision». En: *Universidad Nacional de La Plata Facultad de Informática* (2021).
- [16] Hearst; M.A. et al. «Support vector machines». En: *IEEE Intelligent Systems and their Applications* (1998).
- [17] Ross Girshick. «Fast R-CNN». En: *Computer Vision Foundation* (2015).
- [18] Shaoqing Ren; Kaiming He; Ross Girshick; Jian Sun. «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks». En: *Advances in neural information processing systems* (2015).
- [19] Shahid Karim; Ye Zhang; Shoulin Yin; Irfana Bibi; Ali Anwar Brohi. «A brief review and challenges of object detection in optical remote sensing imagery». En: *Multiagent and Grid Systems An International Journal* (2020).
- [20] Aditya Lohia; Kalyani Dhananjay Kadam; Rahul Raghvendra Joshi; Dr. Anupkumar M. Bongale. «Bibliometric Analysis of One-stage and Two-stage Object Detection». En: *University of Nebraska - Lincoln* (2021).
- [21] Joseph Redmon; Santosh Divvala; Ross Girshick; Ali Farhadi. «You Only Look Once: Unified, Real-Time Object Detection». En: *Computer Vision and Pattern Recognition* (2015).
- [22] Wei Liu; Dragomir Anguelov; Dumitru Erhan; Christian Szegedy; Scott Reed; Cheng-Yang Fu; Alexander C. Berg. «SSD: Single Shot MultiBox Detector». En: *Computer Vision and Pattern Recognition* (2015).
- [23] Tsung-Yi Lin; Priya Goyal; Ross Girshick; Kaiming He; Piotr Dollár. «Focal Loss for Dense Object Detection». En: *Computer Vision and Pattern Recognition* (2017).
- [24] Bichen Wu; Alvin Wan; Forrest Iandola; Peter H. Jin; Kurt Keutzer. «SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving». En: *Computer Vision and Pattern Recognition* (2016).
- [25] Zoumana Keita. *YOLO Object Detection Explained*. <https://www.datacamp.com/blog/yolo-object-detection-explained>. Sep. de 2022. (Visitado 01-04-2024).
- [26] Roboflow. *Roboflow*. <https://docs.roboflow.com/>. 2024.
- [27] Terry Yue Zhuo; Zhou Yang; Zhensu Sun; Yufei Wang; Li Li; Xiaoning Du; Zhenchang Xing; David Lo. «Source Code Data Augmentation for Deep Learning: A Survey». En: *IEEE Intelligent Systems and their Applications* (1998).
- [28] Papers with code. *L2 Regularization*. <https://paperswithcode.com/task/l2-regularization/>. 2024.
- [29] Nitish Srivastava; Geoffrey Hinton; Alex Krizhevsky; Ilya Sutskever; Ruslan Salakhutdinov. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». En: *Journal of Machine Learning Research 15* (2014).
- [30] streamlit. *streamlit*. <https://streamlit.io/>. 2024.

- [31] Mohammed Bany Muhammad; Mohammed Yeasin. «Eigen-CAM: Class Activation Map using Principal Components». En: *Computer Vision and Pattern Recognition* (2020).