
# UNIVERSIDAD TÉCNICA DEL NORTE

## FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

## CARRERA DE SOFTWARE

**INFORME FINAL DEL TRABAJO DE INTEGRACIÓN CURRICULAR,**

**MODALIDAD TRABAJO DE GRADO**

**TEMA:**

# "DESARROLLO DE UNA APLICACIÓN WEB PARA LA DETECCIÓN AUTOMÁTICA DE MALEZAS UTILIZANDO IMÁGENES ADQUIRIDAS POR UN DRON EN CAMPOS DE CULTIVOS DE MAÍZ Y/O PAPA MEDIANTE LA RED PROFUNDA TRANSFORMERS"

**Trabajo de titulación previo a la obtención del título de INGENIERO EN SOFTWARE**

**Línea de investigación:** Desarrollo, aplicación de software y cyber security (seguridad cibernética).

**AUTOR:**

**Luis Armando Perugachi Chávez**

**DIRECTOR:**

**PhD. Iván Danilo García Santillán**

**Ibarra, enero 2025**

**UNIVERSIDAD TÉCNICA DEL NORTE**

**BIBLIOTECA UNIVERSITARIA**

## 1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

| **DATOS DE CONTACTO** | | | |
|---|---|---|---|
| **CÉDULA DE IDENTIDAD:** | 1004796395 | | |
| **APELLIDOS Y NOMBRES:** | Perugachi Chávez Luis Armando | | |
| **DIRECCIÓN:** | Cotacachi, comunidad de Tunibamba – vía Imantag. | | |
| **EMAIL:** | laperugachic@utn.edu.ec, luisprg05@gmail.com | | |
| **TELEFONO FIJO:** | XXXXX | **TELF. MÓVIL:** | 0978926595 |

| **DATOS DE LA OBRA** | |
|---|---|
| **TÍTULO:** | Desarrollo de una aplicación web para la detección automática de malezas utilizando imágenes adquiridas por un dron en campos de cultivos de maíz y/o papa mediante la red profunda Transformers. |
| **AUTOR (ES):** | Perugachi Chávez Luis Armando |
| **FECHA:** | 06/01/2025 |
| SOLO PARA TRABAJOS DE GRADO | |
| **CARRERA/PROGRAMA:** | ■ GRADO ☐ POSTGRADO |
| **TÍTULO POR EL QUE SE OPTA:** | Ingeniero en Software |
| **DIRECTOR:** | PhD. Iván Danilo García Santillán |
| **ASESOR:** | PhD. Marco Remigio Pusda Chulde |

## 2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 6 días del mes de enero del 2025

**EL AUTOR:**

(f)……
Perugachi Chávez Luis Armando

# CERTIFICACIÓN DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 06 de enero de 2025,

PhD. Iván Danilo García Santillán

DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

**CERTIFICA:**

Haber revisado el presente informe final del trabajo de Integración Curricular, mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

IVAN DANILO GARCIA SANTILLAN

Digitally signed by IVAN DANILO GARCIA SANTILLAN
Date: 2025.01.06 12:09:00 -05'00'

(f) .....

PhD. Iván Danilo García Santillán

C.C.: 1002292603

## DEDICATORIA

*Dedico el presente trabajo:*

*A mis padres, especialmente a mi querida madre, María Dolores Chávez Liquinchana, quien fue la única persona que me apoyó en toda mi formación académica. Gracias a su gran sabiduría, me enseñó todos los valores humanos que han definido la persona que soy.*

*Perugachi Chávez Luis Armando*

**AGRADECIMIENTO**

*Manifiesto mi más sincero agradecimiento:*

*A mis padres, María Chávez y Tarquino Perugachi, quienes me han demostrado que con esfuerzo y sacrificio se puede alcanzar cualquier meta. Gracias a ellos, he logrado concluir esta etapa de mi vida.*

*A mis hermanos, Maite, Tarquino y David, quienes siempre han estado a mi lado. Gracias por su invaluable apoyo en la recolección de imágenes para este proyecto.*

*A mi prima Lourdes Perugachi y mi tía Lucía Orbe, gracias por motivarme a seguir luchando por mis sueños con sus invaluables consejos llenas de sabiduría.*

*A mi director, PhD. Iván García, y asesor, PhD. Marco Pusda, por su guía y apoyo profesional para el desarrollo del presente trabajo.*

*A mis mejores amigos, Jhojan y Lizbeth, con quienes he compartido momentos inolvidables, tanto de alegrías como de tristezas. Quienes siempre han confiado en mi a pesar de las adversidades y me han mostrado una amistad sincera y duradera. Gracias por formar parte de mi vida.*

*Perugachi Chávez Luis Armando*

## RESUMEN

La detección y erradicación de malezas es una parte fundamental para mejorar la producción agrícola. Las malezas pueden generar grandes pérdidas económicas al consumir los recursos destinados para los cultivos. Estas malezas comparten bastante similitud con el cultivo y su población suele ser aleatorio en las etapas tempranas de crecimiento. Estos factores influyen directamente en la cuantificación precisa de las mismas, lo que provoca un uso indebido de herbicidas y deterioran significativamente el medio ambiente.

El presente trabajo empleó la arquitectura Real-Time Detection Transformer (RT-DETR) para la detección automática de malezas (kikuyo, diente de león, lengua de vaca y otras malezas no identificadas) en los campos de cultivo de papa. La adquisición del dataset, el entrenamiento y validación del modelo se realizó bajo la metodología KDD. El dataset fue capturado a una altura de 9-10 metros con el dron DJI Mavic 2 Pro a una velocidad de 1 m/s para garantizar una cobertura más amplia del campo de cultivo. La arquitectura RT-DETR fue seleccionada, entre los demás modelos Transformers de la comunidad, por su gran capacidad de detección de objetos en tiempo real, que rivaliza con las famosas redes convolucionales YOLO. El mejor modelo se desplegó en Microsoft Azure a través de una aplicación web empaquetada con Docker.

La evaluación del modelo se llevó a cabo mediante las métricas de rendimiento y la validación estadística. El mejor modelo consiguió 0.8092 mAP y 0.4482 mAP@50:95 en el conjunto de datos de validación (validation set). En la tarea de detección del cultivo y malezas (sin la clase “otros”) se consiguió una precisión media (Average Precision) que oscilan entre 0.8 y 0.9 de AP por cada clase. Sin embargo, el modelo presentó dificultades para identificar las plantas de la categoría “otros”, con una puntuación de 0.528 AP. En cuanto a los resultados de la prueba t-Student, se demostró que no hay una diferencia significativa entre las etiquetas anotadas manualmente (ground truth) y predicciones del modelo en la mayoría de las clases. Estos resultados indican que el modelo es bastante confiable, puesto que, generaliza bien para la inferencia de datos nunca antes vistos.



**Palabras clave:** Deep Learning, agricultura, detección de objetos, detección en tiempo real, detección de malezas, red profunda Transformer, RT-DETR, aplicación web.

## ABSTRACT

Weed detection and eradication is a fundamental aspect of improving agricultural production. Weeds can cause significant economic losses by consuming resources intended for crops. These weeds share a high degree of similarity with the crop and their population is often random in the early stages of growth. These factors directly influence the precise quantification of weeds, leading to the misuse of herbicides and significantly deteriorating the environment.

This work employed the Real-Time Detection Transformer (RT-DETR) architecture for the automatic detection of weeds (kikuyu, dandelion, broadleaf dock, and other unidentified weeds) in potato fields. The dataset acquisition, model training, and validation were conducted under the KDD methodology. The dataset was captured at a height of 9-10 meters using a DJI Mavic 2 Pro drone at a speed of 1 m/s to ensure broader coverage of the crop field. The RT-DETR architecture was selected among other Transformer models due to its superior real-time object detection capabilities, rivaling the well-known convolutional networks such as YOLO. The best model was deployed on Microsoft Azure through a web application packaged with Docker.

Model evaluation was performed using performance metrics and statistical validation. The best model achieved 0.8092 mAP and 0.4482 mAP@50:95 on the validation set. In the task of detecting crops and weeds (excluding the "other" class), a mean Average Precision (AP) between 0.8 and 0.9 was achieved for each class. However, the model struggled to identify plants in the "other" category, with a score of 0.528 AP. The results of the Student´s t-test showed that there is no significant difference between the manually annotated labels (ground truth) and the model's predictions for most classes. These results indicate that the model is quite reliable, as it generalizes well to infer unseen data.



**Keywords:** Deep Learning, agriculture, object detection, real-time detection, weed detection, Transformer deep network, RT-DETR, web application.

## LISTA DE SIGLAS

**AIFI.** Attention-based Intra-scale Feature Interaction.

**AP.** Average Precision**.**

**API.** Application Programming Interface.

**CCFF.** CNN-based Cross-scale Feature Fusion.

**CNN.** Convolutional Neural Network.

**DETR.** Detection Transformer.

**DINO.** DETR with Improved Denoising Anchor Boxes.

**FNN.** Feed Forward Network.

**FPS.** Frames Per Second.

**GPU.** Graphics Processing Unit.

**IoU.** Intersection over Union.

**KDD.** Knowledge Discovery in Databases.

**mAP.** Mean Average Precision.

**NMS.** Non-Maximum Suppression.

**NLP.** Natural Language Processing.

**RNN.** Recurrent Neural Network.

**RT-DETR.** Real-Time Detection Transformer.

**SGD.** Stochastic Gradient Descent.

**TPU.** Tensor Processing Units.

**YOLO.** You Only Look Once.

# ÍNDICE DE CONTENIDOS

**DEDICATORIA** ..... **V**
**AGRADECIMIENTO** ..... **VI**
**RESUMEN** ..... **VII**
**ABSTRACT** ..... **VIII**
**LISTA DE SIGLAS** ..... **IX**
**ÍNDICE DE CONTENIDOS** ..... **X**
**ÍNDICE DE TABLAS** ..... **XIV**
**ÍNDICE DE FIGURAS** ..... **XVI**
**INTRODUCCIÓN** ..... **1**
**Planteamiento del problema** ..... **1**
**Objetivos** ..... **2**
Objetivo General ..... 2
Objetivos Específicos ..... 2
**Alcance** ..... **2**
**Justificación** ..... **3**
**CAPÍTULO I** ..... **5**
**MARCO TEÓRICO** ..... **5**
**1.1. Malezas en la agricultura** ..... **5**
1.1.1. Control de malezas ..... 5
1.1.2. Malezas comunes en los cultivos de papa y maíz ..... 6
**1.2. Arquitectura Transformer** ..... **7**
1.2.1. Bloques codificador y decodificador ..... 8
1.2.2. Arquitectura Detection Transformer (DETR) ..... 9
1.2.3. Arquitectura Real-Time Detection Transformer (RT-DETR) ..... 11
**1.3. Metodología KDD** ..... **12**
1.3.1. Selección ..... 13
1.3.2. Preprocesamiento ..... 14

1.3.3. Transformación ..... 14
1.3.4. Minería de datos ..... 14
1.3.5. Evaluación e interpretación ..... 15
**1.4. Herramientas de desarrollo ..... 15**
1.4.1. Python ..... 15
1.4.2. Tensorflow y Keras ..... 15
1.4.3. PyTorch y Ultralytics ..... 16
1.4.4. Docker ..... 17
1.4.5. Flask ..... 17
1.4.6. React ..... 17
**1.5. ISO 25010 ..... 18**
1.5.1. Característica de Compatibilidad y Portabilidad ..... 19
**1.6. Trabajos relacionados ..... 20**
**CAPÍTULO II ..... 25**
**DESARROLLO ..... 25**
**2.1. Planificación del proyecto ..... 25**
2.1.1. Metodología KDD ..... 25
2.1.2. Instrumentos ..... 27
**2.2. Preparación del dataset ..... 28**
2.2.1. Cultivo de malezas ..... 30
2.2.2. Captura de malezas ..... 32
2.2.3. Selección y preprocesamiento de imágenes ..... 34
2.2.4. Anotación de imágenes ..... 36
2.2.5. Exportación del dataset ..... 38
**2.3. Entrenamiento del modelo Transformer ..... 42**
2.3.1. Selección del modelo Transformer ..... 43
2.3.2. Preparación del entorno del entrenamiento ..... 43
2.3.3. Definición de los hiperparámetros de entrenamiento y aumento de datos ..... 44
2.3.4. Entrenamiento del modelo baseline ..... 46
2.3.5. Experimentación con diferentes optimizadores ..... 47

2.3.6. Experimentación con el modo automatizado de optimización de hiperparámetros 47
2.3.7. Ajuste de hiperparámetros ..... 47
2.3.8. Experimentación con imágenes a diferentes escalas ..... 49
2.3.9. Entrenamiento con el modelo extra large ..... 49
2.3.10. Resumen de entrenamientos ..... 50
**2.4. Desarrollo de la aplicación web ..... 50**
2.4.1. Servidor ..... 51
2.4.2. Cliente ..... 53
2.4.3. Contenedores de desarrollo ..... 55
**2.5. Integración del modelo Transformer en la aplicación web ..... 55**
2.5.1. Empaquetamiento de la aplicación web ..... 56
2.5.2. Despliegue ..... 56
**CAPÍTULO III ..... 57**
**RESULTADOS ..... 57**
**3.1. Métricas de rendimiento ..... 57**
3.1.1. Métricas para detección de objetos ..... 57
3.1.2. Fase 1 – Baseline ..... 58
3.1.3. Fase 2 – Optimizadores ..... 60
3.1.4. Fase 3 – Modo tuneo ..... 64
3.1.5. Fase 4 – Ajuste de hiperparámetros ..... 66
3.1.6. Fase 5 – Escalado de imagen ..... 70
3.1.7. Fase 6 – Modelo extra-large ..... 74
3.1.8. Evaluación del mejor modelo alcanzado ..... 77
**3.2. Validación estadística ..... 88**
3.2.1. Prueba t-Student para la cuantificación de la papa ..... 89
3.2.2. Prueba t-Student para la cuantificación de diente de león ..... 90
3.2.3. Prueba t-Student para la cuantificación del kikuyo ..... 90
3.2.4. Prueba t-Student para la cuantificación de la lengua de vaca ..... 91
3.2.5. Prueba t-Student para la cuantificación de la clase otros ..... 91
3.2.6. Prueba t-Student para la cuantificación de malezas ..... 92

3.2.7. Prueba t-Student para la cuantificación de las plantas ..... 92
3.2.8. Comparativa con las métricas de rendimiento y pruebas t-Student ..... 93
**3.3. Evaluación con la ISO 25010 ..... 94**
3.3.1. Medición de portabilidad ..... 95
3.3.2. Medición de compatibilidad ..... 98
**DISCUSIÓN ..... 101**
**CONCLUSIONES ..... 105**
**RECOMENDACIONES ..... 107**
**REFERENCIAS ..... 109**
**ANEXOS ..... 113**
**Anexo 1 ..... 113**
**Anexo 2 ..... 113**
**Anexo 3 ..... 113**
**Anexo 4 ..... 113**
**Anexo 5 ..... 113**
**Anexo 6 ..... 113**
**Anexo 7 ..... 113**

# ÍNDICE DE TABLAS

Tabla 1 *Listado de malezas comunes* ..... 6
Tabla 2 *Características de las variantes del modelo RT-DETR* ..... 12
Tabla 3 *Características clave de Keras* ..... 16
Tabla 4 *Definición de las subcaracterísticas de compatibilidad y portabilidad* ..... 20
Tabla 5 *Modelos e hiperparámetros* ..... 21
Tabla 6 *Resultados de rendimiento para la detección de malezas y maíz* ..... 23
Tabla 7 *Planificación de las actividades del proyecto* ..... 25
Tabla 8 *Características del dron DJI Mavic 2 Pro* ..... 27
Tabla 9 *Recursos de Kaggle* ..... 28
Tabla 10 *Plantas que conforman el dataset* ..... 29
Tabla 11 *Resumen de visitas al campo de malezas* ..... 33
Tabla 12 *Preprocesamiento de Roboflow* ..... 38
Tabla 13 *Resumen de los datasets obtenidos* ..... 42
Tabla 14 *Hiperparámetros del modelo baseline* ..... 44
Tabla 15 *Hiperparámetros para el aumento de datos* ..... 46
Tabla 16 *Ajuste de hiperparámetros sin considerar la tasa de aprendizaje y el optimizador* ..... 48
Tabla 17 *Resumen de entrenamientos realizados por cada versión del dataset* ..... 50
Tabla 18 *Rutas implementadas en el lado del servidor* ..... 51
Tabla 19 *Parámetros para ajustar los resultados de inferencia* ..... 51
Tabla 20 *Contenidos del fichero comprimido enviado por el servidor* ..... 52
Tabla 21 *Métricas del modelo baseline* ..... 59
Tabla 22 *Métricas de rendimiento bajo distintos optimizadores para la segunda versión del dataset* ..... 60
Tabla 23 *Métricas de rendimiento bajo distintos optimizadores para la tercera versión del dataset* ..... 62
Tabla 24 *Métricas de rendimiento bajo distintos optimizadores para la cuarta versión del dataset* ..... 63
Tabla 25 *Métricas de rendimiento del ajuste de hiperparámetros de la segunda versión del dataset* ..... 66

Tabla 26 *Métricas de rendimiento del ajuste de hiperparámetros de la cuarta versión del dataset* ..... 68
Tabla 27 *Métricas de rendimiento del escalado de imagen de la segunda versión del dataset* ... 71
Tabla 28 *Métricas de rendimiento del escalado de imagen de la cuarta versión del dataset* ..... 73
Tabla 29 *Métricas de rendimiento del modelo extra large de la segunda versión del dataset* .... 75
Tabla 30 *Métricas de rendimiento del modelo extra large de la cuarta versión del dataset* ..... 76
Tabla 31 *Mejores métricas de los dataset versión 2 y 4* ..... 77
Tabla 32 *Métricas de rendimiento de los mejores modelos basados en mAP y mAP@50:95* ..... 79
Tabla 33 *Métricas del mejor modelo RT-DETR alcanzado* ..... 87
Tabla 34 *Resultados de la prueba t-Student para la cuantificación de la papa* ..... 89
Tabla 35 *Resultados de la prueba t-Student para la cuantificación de diente de león* ..... 90
Tabla 36 *Resultados de la prueba t-Student para la cuantificación del kikuyo* ..... 90
Tabla 37 *Resultados de la prueba t-Student para la cuantificación de la lengua de vaca* ..... 91
Tabla 38 *Resultados de la prueba t-Student para la cuantificación de la clase otros* ..... 92
Tabla 39 *Resultados de la prueba t-Student para la cuantificación de malezas* ..... 92
Tabla 40 *Resultados de la prueba t-Student para la cuantificación de las plantas* ..... 93
Tabla 41 *Resultados de las métricas de rendimiento y pruebas t-Student* ..... 93
Tabla 42 *Métricas y fórmulas aplicadas en la medición de portabilidad* ..... 95
Tabla 43 *Resultados de la evaluación de subcaracterística de adaptabilidad* ..... 96
Tabla 44 *Resultados de la evaluación de subcaracterística de facilidad de instalación* ..... 97
Tabla 45 *Resultados de la evaluación de subcaracterística de facilidad de reinstalación* ..... 97
Tabla 46 *Cálculo general del nivel de portabilidad de la aplicación web* ..... 98
Tabla 47 *Métricas y fórmulas aplicadas en la medición de compatibilidad* ..... 99
Tabla 48 *Resultados de la evaluación de subcaracterística de coexistencia* ..... 100
Tabla 49 *Comparativa de modelos Transformer para detección de cultivos y malezas* ..... 102

## ÍNDICE DE FIGURAS

Figura 1 *Árbol de problemas* ..... 1
Figura 2 *Flujo de desarrollo del modelo de inteligencia artificial* ..... 3
Figura 3 *Rábano (Raphanus raphanistrum L.) en diferentes etapas de crecimiento* ..... 7
Figura 4 *Arquitectura Transformer* ..... 8
Figura 5 *Arquitectura de la red Detection Transformer (DETR)* ..... 10
Figura 6 *Arquitectura de la red RT-DETR* ..... 11
Figura 7 *Metodología KDD* ..... 13
Figura 8 *Organización de las series de Normas Internacionales SQuaRE* ..... 18
Figura 9 *Modelo de calidad de producto* ..... 19
Figura 10 *Número de clases obtenidas en el dataset incompleto* ..... 30
Figura 11 *Diente león* ..... 31
Figura 12 *Siembra del diente de león* ..... 31
Figura 13 *Cuidados realizados al campo de malezas* ..... 32
Figura 14 *Ejemplar del campo de malezas* ..... 34
Figura 15 *Muestra de las imágenes seleccionadas y recortadas a 1920x1920 píxeles* ..... 35
Figura 16 *Pantalla de anotación de Roboflow* ..... 36
Figura 17 *Detalles de la primera versión del dataset* ..... 37
Figura 18 *Estructura de directorios YOLOv8* ..... 39
Figura 19 *Segunda versión del dataset de malezas* ..... 40
Figura 20 *Tercera versión del dataset de malezas* ..... 41
Figura 21 *Cuarta versión del dataset* ..... 41
Figura 22 *Sección analizar imagen de la aplicación web* ..... 53
Figura 23 *Sección resultados de la aplicación web* ..... 54
Figura 24 *Sección historial de la aplicación web* ..... 55
Figura 25 *Métrica mAP del modelo baseline* ..... 60
Figura 26 *Métrica mAP bajo distintos optimizadores de la versión 2 del dataset* ..... 61
Figura 27 *Métrica mAP bajo distintos optimizadores de la versión 3 del dataset* ..... 63
Figura 28 *Métrica mAP bajo distintos optimizadores de la versión 4 del dataset* ..... 64

Figura 29 *Métrica mAP del modo de optimización automático de hiperparámetros* ..... 65
Figura 30 *Métrica mAP de los modelos baseline, fase 2 y 4 de la versión 2 del dataset* ..... 68
Figura 31 *Métrica mAP de los modelos baseline, fase 2 y 4 de la versión 4 del dataset* ..... 70
Figura 32 *Métrica mAP de los modelos baseline, fase 2 y 5 de la versión 2 del dataset* ..... 72
Figura 33 *Métrica mAP de los modelos baseline, fase 4 y 5 de la versión 4 del dataset* ..... 74
Figura 34 *Métrica mAP de los modelos baseline, extra-large y el mejor modelo de la versión 2 del dataset* ..... 75
Figura 35 *Métrica mAP de los modelos baseline, extra-large y el mejor modelo de la versión 4 del dataset* ..... 77
Figura 36 *Curva de precision-confidence de modelo best-1* ..... 80
Figura 37 *Curva de precision-confidence de modelo best-2* ..... 80
Figura 38 *Curva de recall-confidence de modelo best-1* ..... 81
Figura 39 *Curva de recall-confidence de modelo best-2* ..... 81
Figura 40 *Curva de precision-recall de modelo best-1* ..... 82
Figura 41 *Curva de precision-recall de modelo best-2* ..... 83
Figura 42 *Matriz de confusión del modelo best-1* ..... 84
Figura 43 *Matriz de confusión normalizada del modelo best-1* ..... 84
Figura 44 *Matriz de confusión del modelo best-2* ..... 85
Figura 45 *Matriz de confusión normalizada del modelo best-2* ..... 86
Figura 46 *Muestras de cultivo y malezas reales, anotadas y predichas por el modelo RT-DETR* 88
Figura 47 *Muestra de etiquetas reales y predicciones negativas del conjunto de datos de testeo* 94

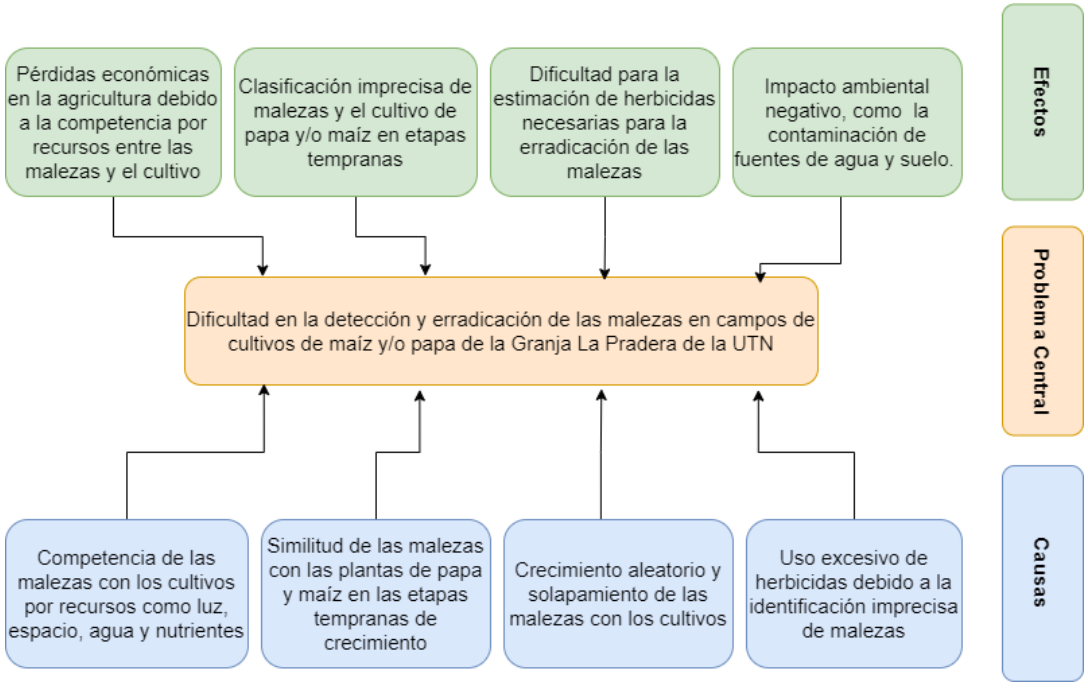
# INTRODUCCIÓN

## Planteamiento del problema

La agricultura es la actividad que más se ha visto afectada por la presencia de malezas, su ineficiente control puede resultar en una reducción drástica de la producción de los cultivos. Las malezas pueden competir con los cultivos por recursos como la luz, agua, espacio y nutrientes (Simón et al., 2018a). En etapas tempranas de algunos cultivos, la vegetación es demasiado parecido a las malezas que los rodean (Thorp & Tian, 2004), además, son difíciles de detectar debido a su crecimiento aleatorio y su solapamiento con el cultivo (Kavir & Delgado, 2021). Estas son algunas de las causas existentes en la Granja Experimental La Pradera de la Universidad Técnica del Norte para que no se tomen medidas de detección y erradicación eficientes en contra de las malezas. En su lugar, se opta por una estimación subjetiva e imprecisa de las malezas. Esto puede provocar un impacto negativo hacia la salud humana y el medio ambiente, debido al uso incorrecto o excesivo de los productos químicos.

**Figura 1**

*Árbol de problemas*




*Fuente:* Autoría propia.

## Objetivos

### Objetivo General

Desarrollar una aplicación web para la detección automática de malezas utilizando imágenes adquiridas por un dron en campos de cultivos de maíz y/o papa mediante la red profunda Transformers.

### Objetivos Específicos

- Redactar un marco teórico sobre la detección de malezas en cultivos basado en la arquitectura de red Transformers.
- Entrenar una red Transformers para clasificar malezas en cultivos de papa y/o maíz utilizando el Framework Tensorflow y la librería Keras.
- Desarrollar una aplicación web para el despliegue del modelo entrenado que cumpla con las características de compatibilidad y portabilidad de la ISO 25010.
- Evaluar los resultados del modelo entrenado frente a un conjunto de datos reales, utilizando métricas de inteligencia artificial y estadística descriptiva.

## Alcance

El alcance del presente trabajo se enmarca en la investigación y desarrollo de una aplicación web que implemente la red profunda de tipo Transformers (Vaswani et al., 2017) para la detección de malezas presentes en los cultivos de papa y/o maíz usando imágenes adquiridas por un dron bajo la metodología KDD. Para ello, se preparará un repositorio con al menos 100 imágenes de campos de cultivo de maíz y/o papa en la granja La Pradera de la Universidad Técnica del Norte. Estas imágenes serán capturadas por el dron DJI Mavic 2 Pro y serán etiquetadas para identificar el cultivo y la maleza.

La arquitectura de red profunda Transformers fue seleccionada por su capacidad y versatilidad que se ha extendido a aplicaciones más allá del procesamiento del lenguaje natural, incluyendo la visión por computadora y otros campos de la inteligencia artificial. Transformers es el primer modelo de transducción de secuencia basado completamente en mecanismos de atención (Vaswani et al., 2017).

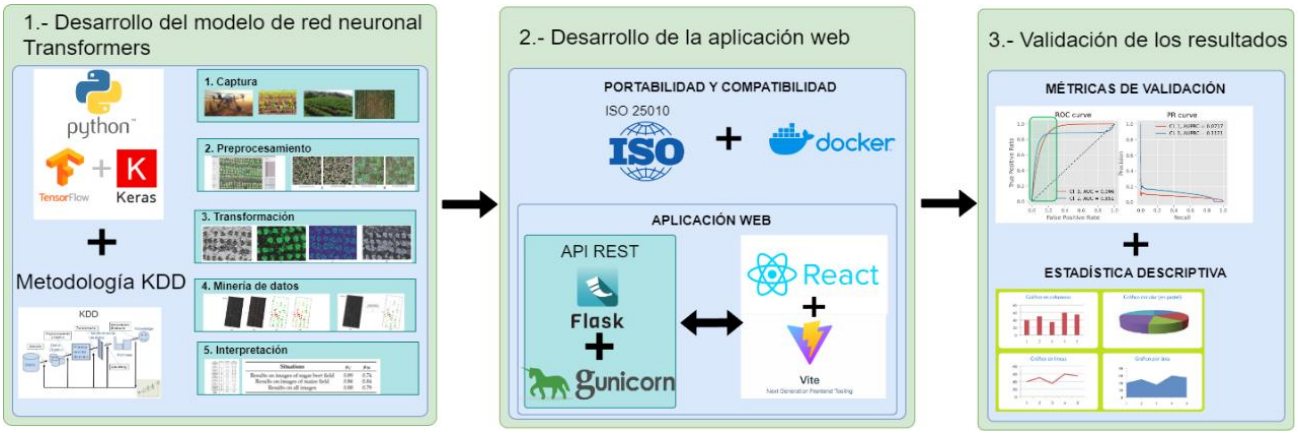
Se utilizará Python como lenguaje de programación, debido a su versatilidad y sencillez a la hora de integrarse con muchos algoritmos de Machine Learning y Deep Learning. El framework para entrenar el modelo será TensorFlow y Keras. Se trabajará con Google Colaboratory para el entrenamiento del modelo.

La aplicación web está basada en una arquitectura Cliente-Servidor. El servidor (Back-End) será construido en el Framework de desarrollo web Flask escrito en Python. Para desarrollar la interfaz de usuario web (Front-End) se usará la librería React que está escrita en JavaScript. Además, se va a implementar contenedores de Docker para asegurar la disponibilidad y compatibilidad de paquetes y librerías usadas en el proyecto, como se muestra en la Figura 2.

Dado que el presente trabajo está enfocado en el entrenamiento de la red Transformers no se va a aplicar una metodología de desarrollo ágil o tradicional. Sin embargo, se va a garantizar la calidad de software mediante las características de compatibilidad y portabilidad de la norma ISO 25010.

**Figura 2**

*Flujo de desarrollo del modelo de inteligencia artificial*




*Fuente:* Autoría propia.

**Justificación**

La presente investigación está alineada con el objetivo doce de los Objetivos de Desarrollo Sostenible (ODS), que busca reducir la liberación de los productos químicos al aire, suelo y agua para disminuir su impacto hacia la salud humana y el medio ambiente (Naciones Unidas, 2016). También, la política 12.1 del objetivo doce del plan de creación de oportunidades 2021 y 2025

hace énfasis en mejorar las acciones de mitigación y adaptación al cambio climático (SENPLADES, 2021).

**Justificación Tecnológica. -** Dentro de la política agropecuaria ecuatoriana, específicamente el capítulo 3.1.6 expone la falta de investigación, desarrollo e innovación tecnológica en la agricultura destacando que no hay suficientes procesos de innovación tecnológica y social, que permitan mejorar los rendimientos agroproductivos en los cultivos (MAGAP, 2016).

**Justificación Ambiental. -** En el plan de creación de oportunidades 2021-2025, el objetivo 11 menciona la importancia de conservar, restaurar, proteger y hacer uso racional de los recursos naturales a través de la implementación de mejores prácticas ambientales (SENPLADES, 2021).

Bajo esta perspectiva, se tiene a los agricultores como los principales beneficiarios del proyecto; además, la salud de los consumidores y el medio ambiente serán los beneficiarios indirectos.

# CAPÍTULO I

# MARCO TEÓRICO

## 1.1. Malezas en la agricultura

Las malezas son plantas que crecen aleatoriamente en los campos de cultivo, y que interfieren directamente con la cantidad de producción agrícola que se pueda obtener. Estas plantas compiten con los cultivos por los recursos esenciales como la luz solar, el agua y los nutrientes del suelo, lo que puede resultar en un crecimiento reducido y una menor producción de los cultivos (Simón et al., 2018b). Sin embargo, también provocan otros efectos perjudiciales sobre los cultivos, por ejemplo, la contaminación de los productos cosechados que afectan directamente su valor comercial. La presencia de malezas entorpece la cosecha, incrementa el costo de limpieza de los productos cosechados y eleva la probabilidad de que los campos de cultivos se infecten de plagas y enfermedades (Simón et al., 2018a).

### 1.1.1. Control de malezas

El control de malezas se realiza con el propósito de mitigar la población de maleza que se encuentran presentes en los campos de cultivo. Existen diferentes estrategias de control preventivo, culturales y curativos para el manejo de malezas. El método preventivo se realiza antes de la siembra del cultivo mientras que los métodos culturales y curativos se aplican en las etapas de crecimiento del cultivo (Bàrberi et al., 2004). Algunos de estos métodos son el arado del suelo, deshierbe manual o la rotación de cultivos, sin embargo, el control químico es uno de los más comunes debido a su efectividad y se aplica en las diversas etapas del cultivo.

Durante la primera etapa, también conocida como “barbecho químico”, los herbicidas se aplican en el periodo de “barbecho” del campo. La segunda etapa se conoce como “presiembra incorporada”. El control se realiza mediante la aplicación de las herbicidas en el estado previo a la siembra del cultivo con métodos como la labranza para que el suelo se mezcle físicamente con los herbicidas. La siguiente etapa se denomina “preemergencia”, donde se aplican herbicidas justo antes de que el cultivo o las malezas emerjan del suelo. Finalmente, en la etapa de

“postemergencia” se aplican directamente herbicidas en las malezas visibles que han emergido del suelo (Simón et al., 2018a).

### 1.1.2. Malezas comunes en los cultivos de papa y maíz

A continuación, se listan las malezas comunes que se pueden encontrar en los campos de cultivo de papa y maíz:

**Tabla 1**

*Listado de malezas comunes*

| Nombre científico | Nombre común | Nivel de persistencia | Tipo de reproducción |
|---|---|---|---|
| Taraxacum officinale | Diente de león | Media | Semillas |
| Rumex crispus | Lengua de vaca | Alta | Semillas, rizomas |
| Amaranthus hibridus | Bledo | Alta | Semillas |
| Galisonja parviflora | Guasca | Baja | Semillas |
| Malva preuvianum | Malva blanca | Media | Semillas |
| Senecio | Escobilla | Alta | Semillas |
| Pennisetum clandestinum | Kikuyo | Alta | Semilla, estolones |
| Raphanus raphanistrum | Rábano | Alta | Semillas, raíz |
| Trifolium repens | Trébol blanco | Alta | Semillas, estolones |
| Brassica spp. | Nabo, Alpiste | Alta | Semilla |

*Fuente:* Adaptada de (Arrieta, 2000)

Como se observa en la Tabla 1, las malezas se clasifican por su nivel de persistencia y su tipo de reproducción, por ejemplo, el rábano (Raphanus raphanistrum) tiene el nivel de persistencia alto, lo que indica que la maleza tiene una gran capacidad de adaptabilidad a los diferentes entornos y etapas de crecimiento como se muestra en la Figura 3 (Arrieta, 2000). Por lo tanto, las malezas con el nivel de persistencia alto deben tener una mayor prioridad en su eliminación para disminuir su población.

**Figura 3**

*Rábano (Raphanus raphanistrum L.) en diferentes etapas de crecimiento*

*Nota.* La figura muestra el crecimiento del rábano en un cultivo de papas. Tomado de (Torres, 2018).

## 1.2. Arquitectura Transformer

Las redes profundas Transformer surgen a partir de las limitaciones que presentan las redes neuronales recurrentes (RNN) para el análisis del procesamiento de lenguaje natural (NLP). Las RNNs memorizan eficientemente la información procesada, capa tras capa, de manera secuencial (Hopfield, 1982). Estas redes se basan en la forma intuitiva que tiene el ser humano para procesar la información escrita, lo que las hace muy útiles para la resolución de problemas de procesamiento de lenguaje natural, como la traducción de texto o la generación de texto.

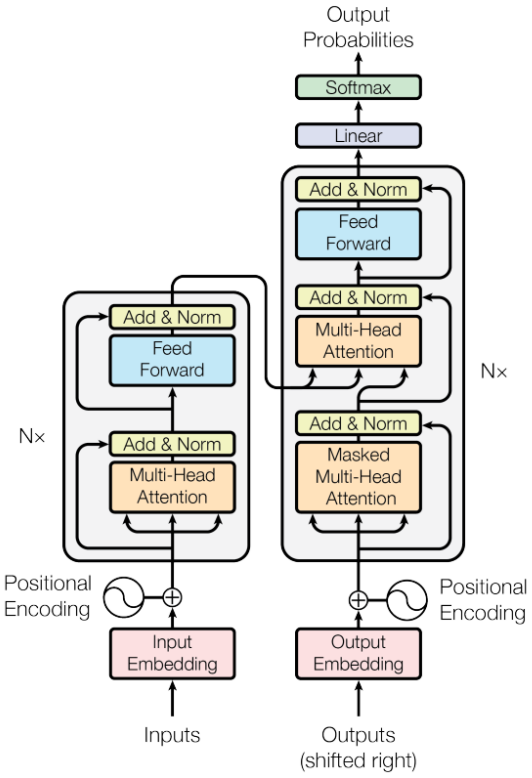
A pesar de los beneficios que las RNNs puedan ofrecer, presentan limitaciones muy notables a la hora de trabajar con grandes cantidades de información. Dado a la naturaleza secuencial de las RNNs, es difícil mantener el peso que tienen la información procesada al principio de la red neuronal frente a la información al final de la red (Vaswani et al., 2017). Este problema puede interpretarse como si la red estuviera "olvidando información". Otro

inconveniente que se presenta es la dificultad para paralelizar el entrenamiento de la red. Esto se debe a que las palabras de una frase son procesadas secuencialmente, es decir, cada palabra depende directamente de la información procesada de la palabra anterior para ser procesada. Al ser un problema secuencial, es difícil procesar cada palabra de forma individual y al mismo tiempo, lo que provoca que se pierda las prestaciones que puedan ofrecer las GPUs.

Como se observa en la Figura 4, la arquitectura se divide en el bloque del codificador, que está en la parte la izquierda, y el bloque decodificador ubicado a la derecha de la figura.

**Figura 4**

*Arquitectura Transformer*




*Nota.* La figura muestra la arquitectura Transformer presentada por primera vez en 2017. Tomada de (Vaswani et al., 2017).

### 1.2.1. Bloques codificador y decodificador

Los bloques codificador y decodificador que se presenta en la Figura 4, tienen similitudes entre sí, ambos bloques tienen seis capas en total e implementan mecanismos de atención.

Los mecanismos de atención son operaciones matemáticas realizadas sobre conjunto de datos para prestar atención a partes específicas del mismo. La arquitectura Transformer se basa completamente en mecanismos de atención, donde se usa self-attention (scaler dot-product attention) y multi-head attention (Vaswani et al., 2017). Self-attention permite que los vectores de entrada decidan cuanta atención prestar al resto de vectores de forma autónoma. En cambio, multi-head attention usa self atttention y lo repite un número determinado de veces para mejorar la capacidad de representación de datos (Sensio, 2021).

El codificador divide las 6 capas en 2 subcapas, la primera subcapa implementa un mecanismo de atención multi-head, este mecanismo lee los vectores query, key y values para alimentar a la siguiente capa mediante un procesamiento en paralelo. La segunda capa es una red conectada que usa 2 transformaciones lineales con la activación de Unidad Lineal Rectificada (ReLU) (Brownlee et al., 2022).

El decodificador se divide en tres subcapas. En la primera subcapa, a diferencia del codificador, que procesa todas las entradas sin importar su posición, esta se encarga solo de tratar las entradas anteriores con el mismo mecanismo de atención multi-head. La segunda subcapa atiende la salida de la subcapa anterior del decodificador y la salida del codificador, es decir, las keys y values (K, V) (Rothman, 2022). Como se muestra en la ecuación 1, la última subcapa es similar a la subcapa implementada en el codificador, pero esta se prepara para recibir la información del codificador.

$$Input\_Attention = (Output\_decoder\_sub\_layer - 1(Q), Output\_encoder\_layer(K, V)) \quad (1)$$

*Fuente.* Tomado de (Rothman, 2022).

Para el correcto funcionamiento de la arquitectura Transformers, se hace uso de la codificación posicional, la cual captura la posición de las entradas alimentadas a esta red, dado que los bloques descritos anteriormente carecen de la habilidad de almacenar información posicional.

### 1.2.2. Arquitectura Detection Transformer (DETR)

Una de las tareas comunes de la visión por computadora es la detección de objetos en una imagen. Normalmente, las arquitecturas basadas en CNN han demostrado buenos resultados al abordar este problema. Sin embargo, desde la presentación de la arquitectura Transformer, el

desarrollo de nuevas arquitecturas basadas en esta tecnología ha continuado sin pausa para resolver tareas distintas al procesamiento de lenguaje natural.

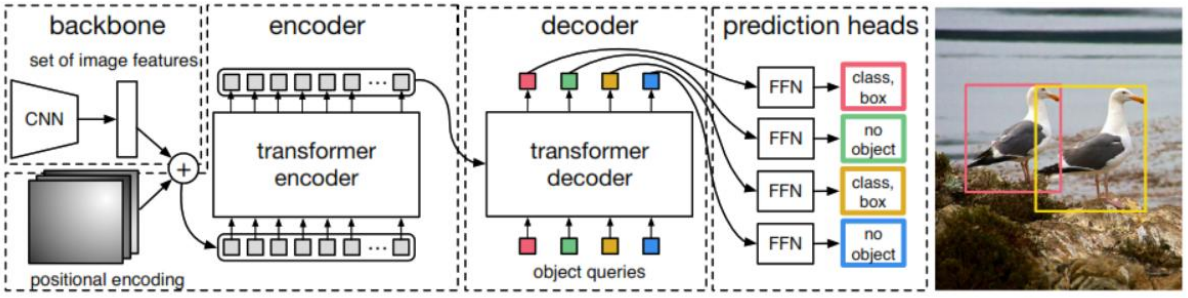
DETR es una arquitectura basada en la red profunda Transformer y una CNN. Como se puede observar en la Figura 5, el papel principal de la CNN es la extracción de características de la imagen, como esquinas, bordes, texturas, etc. Posteriormente, la red Transformer toma las características y las procesa a través de sus mecanismos de atención. Para generar la predicción final, se hace uso de una red Feed Forward Network (FFN) que, en este caso, devuelve como salida la clase y las coordenadas de cuadro delimitador del objeto (Carion et al., 2020).

Esta arquitectura simplifica el proceso de detección de objetos eliminando componentes usados en las CNN, como el anclaje espacial (spatial anchors) o la supresión de no máximos (NMS), para predecir la ubicación de los objetos directamente (Carion et al., 2020). Este tipo de modelos se conoce como detectores de una sola etapa (single-stage models).

De acuerdo a los experimentos realizados por Carion et al., (2020), el modelo DETR-DC5-R101, con 60 millones de parámetros, obtuvo 44.9 de AP, 64.7 de AP50 y 47.7 de AP75. Al compararlo con el modelo Faster RCNN-R101-FPN+, este alcanzó 44 de AP, 63.9 de AP50 y 47.8 de AP75, donde DETR supera ligeramente a Faster R-CNN. Además, DETR mostró mejores resultados al detectar objetos grandes, pero tuvo problemas al identificar objetos pequeños.

**Figura 5**

*Arquitectura de la red Detection Transformer (DETR)*




*Nota.* La figura muestra la arquitectura de DETR divido en los siguientes bloques: backbone, encoder, decoder y prediction heads. Tomada de (Carion et al., 2020).

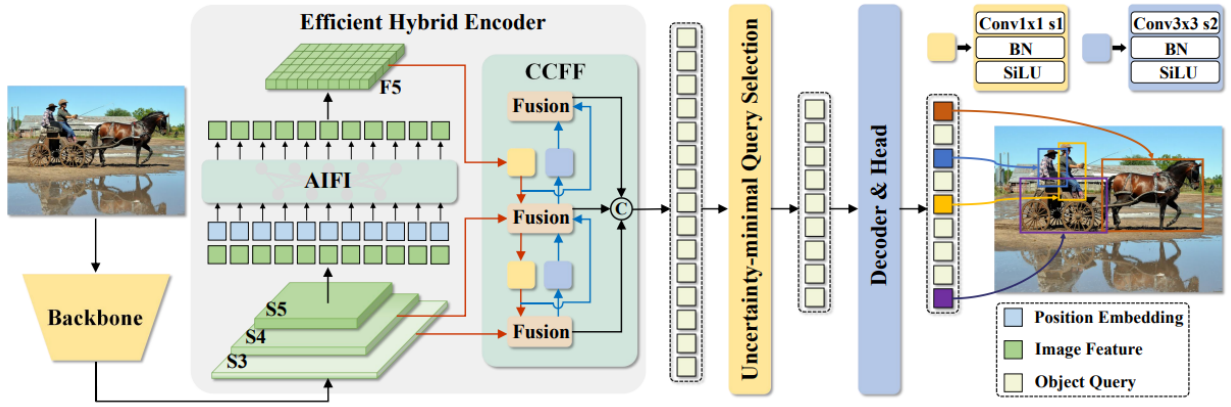
### 1.2.3. Arquitectura Real-Time Detection Transformer (RT-DETR)

Con el éxito de la incorporación de las redes Transformer para la detección de objetos, a partir de DETR se fueron incorporando modificaciones para mejorar la precisión y como una alternativa para eliminar el uso de la supresión de no máximos (NMS). Sin embargo, el alto costo computacional que requería el uso de estos modelos impedía que los modelos basados en Transformer se pudieran aplicarse a la tarea de detección de objetos en tiempo real (Zhao et al., 2024).

La arquitectura RT-DETR fue diseñada para aprovechar las ventajas del modelo DETR y mejorar significativamente en términos de eficiencia computacional y precisión. Las partes principales del modelo son el backbone, el codificador eficiente híbrido y el decodificador con cabezales de predicción auxiliares como se muestra en la Figura 6.

**Figura 6**

*Arquitectura de la red RT-DETR*




*Fuente:* Tomada de (Zhao et al., 2024).

En comparación con el modelo DETR y sus variantes (Deformable-DETR, DINO, etc.), se rediseñó los componentes que causaban un cuello de botella computacional. Se analizó la redundancia computacional que generaba el codificador multiescalar, el cual contiene información semántica de alto nivel de los objetos extraída de las características de bajo nivel (Zhao et al., 2024). Para solucionar este cuello de botella, se propuso un codificador híbrido eficiente (efficient hybrid encoder) compuesto de dos módulos: Attention-based Intra-scale Feature Interaction (AIFI) y CNN-based Cross-scale Feature Fusion (CCFF).

El funcionamiento se resume en que el codificador híbrido eficiente recibe la información extraída de la imagen del backbone en sus tres etapas (S3, S4, S5); luego, el codificador procesa las características multiescala en una sola secuencia a través de los módulos AIFI y CCFF. Seguidamente, el bloque selector de consultas de mínima incertidumbre (uncertainty-minimal query selection) elige un número fijo de características del objeto y las transfiere al decodificar para que, de forma iterativa, prediga las cajas y categorías del objeto. En la Tabla 2 se observa la generalidades de las variantes large y extra-large del modelo RT-DETR.

**Tabla 2**

*Características de las variantes del modelo RT-DETR*

| Model | AP on COCO val2017 | Params | Layers | Frames per second on T4 GPU | Size in MB |
|---|---|---|---|---|---|
| RT-DETR-L | 53.0% | 32,816,351 | 681 | 114 FPS | 64 MB |
| RT-DETR-X | 54.8% | 67,313,727 | 875 | 74 FPS | 130 MB |

*Fuente:* Obtenido de (Jocher et al., 2023).

Los modelos RT-DETR-R50 y RT-DETR-R101 alcanzaron una AP de 53.1% y 54.3%, respectivamente, con velocidades de procesamiento de 108 y 74 FPS. En comparación con YOLOv7-L y YOLOv8-L, RT-DETR-R50 mostró una mejora de 1.9% y 0.2% en AP, y un aumento del 96.4% y 52.1% en FPS. De la misma forma, RT-DETR-R101 superó a YOLOv7-X y YOLOv8-X en un 1.4% y 0.4% en AP, y en un 64.4% y 48.0% en FPS (Zhao et al., 2024). Estos resultados prueban que RT-DETR se encuentra al nivel de los modelos actuales para la detección de objetos en tiempo real.

### 1.3. Metodología KDD

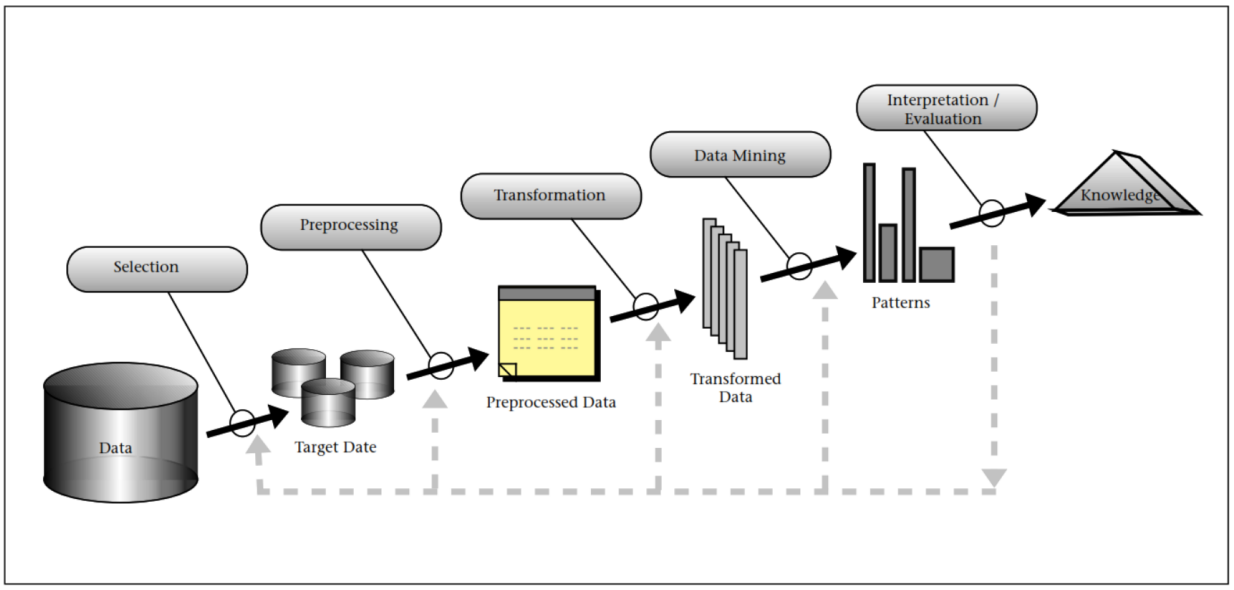
En el mundo globalizado que se vive actualmente, la información que se tiene a disposición es masiva. Este concepto es mejor conocido como “Big Data” en el contexto de la inteligencia artificial. Analizar y reconocer patrones en estas enormes cantidades de datos es un desafío que enfrentan los científicos de datos. Sin embargo, este problema ha estado presente desde varias décadas atrás. Debido a la necesidad de extraer información valiosa, han salido a la luz varias metodologías que han ayudado a tener un marco de trabajo claro a la hora de tratar datos masivos.

Una de las primeras metodologías que salieron a la luz fue KDD (Knowledge Discovery in Databases). Esta metodología es muy popular y ampliamente usada entre los científicos de datos. El principal problema que aborda es el tratamiento de datos masivos en formas más ligeras y abstractas (Fayyad et al., 1996). Es decir, ayuda a interpretar mejor los datos y a reconocer patrones ocultos en la inmensidad de los datos.

Según Fayyad et al., (1996), la metodología KDD es un proceso iterativo donde intervienen varios pasos que el usuario puede adaptar a sus necesidades y seguir el flujo que decida, como se puede observar en la Figura 7.

**Figura 7**

*Metodología KDD*




*Nota.* Pasos que componen la metodología KDD. Tomada de (Fayyad et al., 1996).

### 1.3.1. Selección

De acuerdo con la metodología KDD, la selección de datos implica la creación del conjunto de datos, enfocándose en las variables clave que facilite la minería de datos (data mining) (Fayyad et al., 1996). Sin embargo, para el desarrollo de modelos de Deep Learning es sumamente recomendable adquirir la mayor cantidad de datos posible, ya que esto garantiza que el modelo tenga suficiente cantidad de datos y generalice bien la tarea asignada.

### 1.3.2. Preprocesamiento

Una vez reunido todo el conjunto de datos, según Fayyad et al. (1996), se debe realizar el limpiado y preprocesamiento de datos mediante la aplicación de diferentes estrategias. Por ejemplo, en el caso de tratamiento de imágenes, existen técnicas como reducción de ruido, realce de contraste y la mejora de ciertas características de la imagen (García, 2008). Este paso es sumamente importante, puesto que de esto depende el éxito o fracaso del modelo para aprender a realizar correctamente la tarea asignada.

### 1.3.3. Transformación

La transformación de datos implica modificar los datos ya preprocesados. En el contexto de tratamiento de imágenes, algunas de las estrategias son el aumento de datos (data augmentation), que consiste en incrementar imágenes aplicando diferentes operaciones como la rotación, desplazamiento, cambio de orientación, entre otras. Otra estrategia de transformación de datos es la normalización o estandarización, que ayuda a ajustar los datos a una escala común y fácil de manejar. Por ejemplo, en el manejo de imágenes, se ajusta el rango de valores de pixeles, que originalmente va de 0 y 255 a valores normalizados entre 0 y 1.

### 1.3.4. Minería de datos

Según Fayyad et al., (1996), en este punto se debe escoger un método particular para la minería de datos que solucione los problemas de clasificación, regresión o clustering. Para identificar el algoritmo adecuado, es importante tener una compresión clara y concisa del problema que se está tratando. De acuerdo con el problema que se esté tratando, ya sea de clasificación, regresión o clustering, existen algoritmos especializados para abordar estos problemas.

Otros puntos que se deben considerar a la hora de escoger el algoritmo adecuado son:

- El tamaño del conjunto de datos
- Facilidad de implementación del algoritmo
- Sensibilidad a la escala de datos y valores atípicos
- Tiempo de inferencia
- Capacidad de cómputo disponible

- Tiempo de entrenamiento
- Facilidad de interpretación

### 1.3.5. Evaluación e interpretación

Como se puede observar en la Figura 7, la interpretación y evaluación son el paso final de la metodología KDD, donde se utilizan diferentes métricas para evaluar el modelo. Una vez interpretados los datos minados, siempre existe la posibilidad de regresar a cualquiera de los pasos previos hasta que se logre obtener los resultados deseados (Fayyad et al., 1996).

## 1.4. Herramientas de desarrollo

En la actualidad, el desarrollo de software se encuentra en constante cambio. Muchas de las tareas repetitivas están automatizadas por frameworks y librerías que son impulsadas por lenguajes de programación potentes y sencillos de comprender. Gracias a estas herramientas, es posible enfocar el tiempo y esfuerzo netamente en resolver el problema. A continuación, se revisarán brevemente las herramientas esenciales para el presente trabajo.

### 1.4.1. Python

Python es uno de los lenguajes de programación más importantes en la actualidad. Esto se debe a que permite trabajar rápidamente e integrar sistemas de manera efectiva (Python Software Foundation, 2024). Debido a su legibilidad y facilidad de comprensión, su curva de aprendizaje sea amigable con el estudiante. Para empezar con Python, simplemente hace falta tener instalado el intérprete de Python y un archivo de texto plano con la extensión ".py", con las respectivas instrucciones para que el intérprete lo ejecute.

### 1.4.2. Tensorflow y Keras

Uno de los frameworks principales para la creación de modelos de aprendizaje automático en la actualidad es Tensorflow, éste proporciona un ambiente de trabajo completo para acelerar todo el flujo de trabajo. Tensorflow ofrece herramientas para la preparación de datos, la creación de modelos y la implementación estos en una amplia variedad de las plataformas, como la web, móvil y servidores (Tensorflow Developers, 2023).

Tensorflow proporciona a la comunidad open source repositorios de modelos entrenados, listos para ser implementados. Entre los que se destacan se encuentran Tensorflow Hub y Model Garden. Esta colección de modelos es desarrollada, mantenida y actualizada a las últimas tendencias del mercado por el equipo de Tensorflow. Otro punto importante de Tensorflow es su compatibilidad de integración para integrarse con otros lenguajes de programación, por ejemplo, Tensorflow.js para el entrenar y desplegar de modelos utilizando JavaScript o Node.js. De manera similar, existen implementaciones para java mediante Tensorflow JVM (Tensorflow Developers, 2023).

Keras es la API de alto nivel de TensorFlow para entrenar modelos de aprendizaje profundo. En la nueva versión 3 de Keras, se incorporan mejoras sustanciales y se añaden nuevas características, como el soporte para múltiples backends (TensorFlow, PyTorch y JAX) sin necesidad de cambiar líneas de código, así como mejoras en el rendimiento de entrenamiento e inferencia en GPU, TPU y CPU (Chollet, 2023). Entre sus principales características, destacan tres ventajas fundamentales, como se puede observar en la Tabla 3.

**Tabla 3**

*Características clave de Keras*

| Característica clave | Detalle |
|---|---|
| Amigable con el usuario | Keras brinda una API simple y potente que facilita su compresión |
| Modular y configurable | Cada modelo de Keras se crea como una serie de piezas intercambiables que son altamente configurables |
| Fácil de extender | Es muy flexible a la hora de crear modelos personalizados |

*Fuente:* Adaptada de (Tensorflow Developers, 2023).

### 1.4.3. PyTorch y Ultralytics

PyTorch es una alternativa para entrenar modelos de aprendizaje profundo. Ofrece un flujo de trabajo completo que permite manipular datos, crear modelos, optimizar parámetros y guardar los modelos entrenados (Subramanian et al., 2024). Dentro de la comunidad, se considera una opción que ofrece mayor libertad para crear y personalizar modelos de inteligencia artificial. A

diferencia de Keras, esto hace que su curva de aprendizaje sea más desafiante. No obstante, ofrece todas las funcionalidades disponibles en TensorFlow y Keras.

La librería Ultralytics es ampliamente reconocida por ofrecer soluciones basadas en visión por computador a problemas en agricultura, manufactura, conducción autónoma y salud. Esta biblioteca está construida sobre PyTorch y ofrece una API intuitiva que permite realizar tareas como clasificación de imágenes, detección de objetos y segmentación semántica.

Ultralytics ofrece una amplia gama de modelos preentrenados, entre los cuales destacan los modelos de la serie YOLO. El modo de entrenamiento de Ultralytics destaca por su eficiencia al utilizar los recursos disponibles, como CPU o GPU. Además, la configuración de los hiperparámetros puede realizarse mediante ficheros YAML, argumentos en la línea de comandos o directamente a través del código (Ultralytics, 2024).

### 1.4.4. Docker

Docker es una plataforma ampliamente utilizada en el desarrollo de software para empaquetar y distribuir aplicaciones mediante contenedores. Los contenedores son entornos aislados del sistema operativo anfitrión que operan con su propio núcleo del sistema basado en Linux, eliminando la necesidad de emulación, como ocurre con las máquinas virtuales (Docker, 2023).

### 1.4.5. Flask

Flask es una de las alternativas más populares para desarrollar aplicaciones web. Su curva de aprendizaje es sencilla, y permite crear un servidor web de forma rápida con solo un par de líneas de código (Ronacher, 2023). Flask es un framework flexible, donde el usuario tiene total control para definir la arquitectura de software a implementar.

### 1.4.6. React

En la actualidad, el desarrollo web del lado del cliente se ha visto impulsado gracias a numerosas bibliotecas que automatizan tareas repetitivas. Una de las más destacadas es React, escrita en JavaScript. React se basa en la creación de interfaces a partir de piezas individuales conocidas como componentes (Meta Developers, 2022). Trabajar con React implica un cambio de

paradigma hacia la programación funcional y reactiva, lo que permite que el código sea declarativo y más fácil de comprender para el programador.

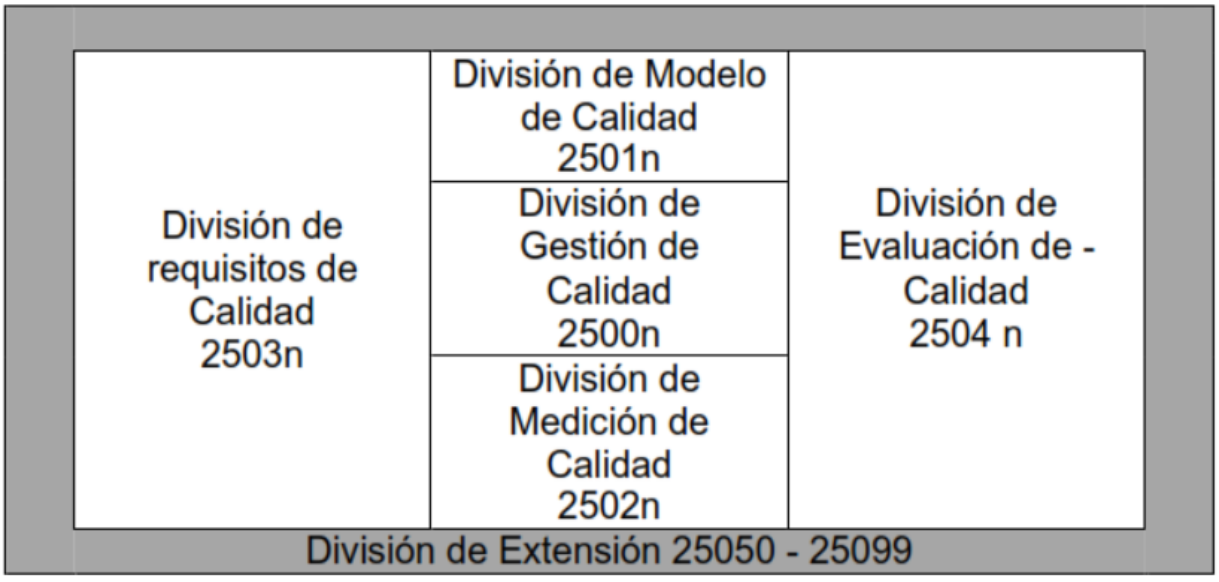
## 1.5. ISO 25010

La calidad del software se logra cumpliendo ciertas características que califican al sistema. Estos atributos son definidos por el equipo de desarrollo para garantizar el cumplimiento de los objetivos requeridos. Uno de los estándares que ayuda a evaluar la calidad del software es la norma ISO/IEC 25010.

La ISO/IEC 25010 forma parte de un grupo de normas internacionales conocido como SQuaRE, específicamente de la división de calidad, como se muestra en la Figura 8. Esta división proporciona directrices para evaluar la calidad del software y puede aplicarse a sistemas de computación, productos de software, calidad de uso y calidad de los datos (Joint Technical Committee, 2011).

**Figura 8**

*Organización de las series de Normas Internacionales SQuaRE*




*Nota.* La figura muestra la división de la familia de normas SQuaRE. Tomado de (Joint Technical Committee, 2011).

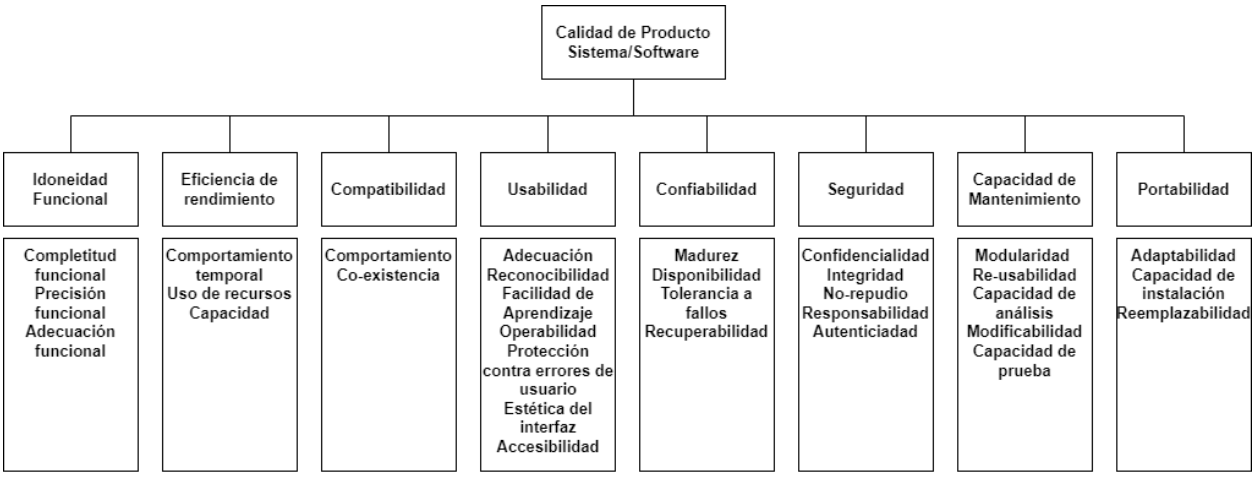
Según Joint Technical Committee. (2011), el estándar de calidad ISO/IEC 25010 proporciona un marco de referencia para el modelo de calidad. Se entiende como calidad el grado

en que se satisfacen las necesidades establecidas por las partes interesadas. Este marco de referencia se divide en tres modelos, de los cuales la ISO/IEC 25010 abarca el modelo de calidad de uso y el modelo de calidad de producto.

El modelo de calidad de producto define ocho características esenciales que el producto debe cumplir; estas características, a su vez, se descomponen en subcaracterísticas, como se ilustra en la Figura 9. El uso de cada una de estas características depende directamente de los objetivos del proyecto en evaluación.

**Figura 9**

*Modelo de calidad de producto*




*Nota.* La figura muestra las características y subcaracterísticas de modelo de calidad de producto. Tomada de (Joint Technical Committee, 2011).

### 1.5.1. Característica de Compatibilidad y Portabilidad

La compatibilidad y la portabilidad son características clave del modelo de calidad de producto y se aplican a un sistema completo. La compatibilidad se define como el nivel de intercambio de información entre un sistema o componente y otro producto, sistema o componente, para realizar sus tareas en un entorno compartido, ya sea de hardware o software. Por otro lado, la portabilidad se entiende como el grado de efectividad y eficiencia con el que un sistema, producto o componente puede transferirse a otra plataforma, software, hardware u otro entorno, y seguir funcionando normalmente (Joint Technical Committee, 2011).

Cada característica incluye subcaracterísticas que contribuyen a su cumplimiento, tal como se detalla en la Tabla 4.

**Tabla 4**

*Definición de las subcaracterísticas de compatibilidad y portabilidad*

| Característica | Subcaracterística | Definición |
|---|---|---|
| Compatibilidad | Coexistencia | Grado en el que varios sistemas puede funcionar en el mismo entorno, sin perjudicarse entre sí |
| | Interoperabilidad | Grado en el que varios sistemas pueden intercambiar información entre sí |
| Portabilidad | Adaptabilidad | Grado en el que un software puede funcionar efectivamente en diferentes entornos |
| | Capacidad de instalación | Grado de efectividad de un sistema para ser instalado y desinstalado en cualquier ambiente |
| | Capacidad de ser reemplazado | Grado en el que un sistema puede reemplazar a otro sistema para el mismo propósito en el mismo entorno |

*Fuente:* Adaptado de (Joint Technical Committee, 2011).

## 1.6. Trabajos relacionados

En el trabajo de Abuhani et al. (2023), se realiza una comparación de cuatro arquitecturas de Deep Learning: YOLO (You Only Look Once), EfficientDet, RetinaNet y DETR (Detection Transformer). Estas arquitecturas se emplearon para identificar malezas en cultivos de remolachas y girasoles. Cada cultivo tiene su propio conjunto de datos con 1,600 ejemplares de remolachas y 500 imágenes de girasoles. Las tres métricas utilizadas para validar los modelos fueron IoU (Intersection over Union), AP (Average Precision) y mAP (Mean Average Precision). Los hiperparámetros utilizados se describen en la Tabla 5. De acuerdo con los resultados obtenidos, el modelo YOLOv7 obtuvo una mAP de 81.6% sobre el data set de remolachas y YOLOv8 obtuvo una mAP de 77 % en data set de girasoles, sin embargo, DETR consiguió una mAP de 35.3% y 43 % para cada data set. Es importante destacar que el estudio no utiliza estrategias para mejorar los

resultados, como Transfer Learning, Data Augmentation o Fine-Tuning, lo cual justificaría los pobres resultados obtenidos por la arquitectura DETR.

**Tabla 5**

*Modelos e hiperparámetros*

| Model | Backbone | Epochs | Optimizer | Learning rate | Batch Size |
|---|---|---|---|---|---|
| YOLOv8 | E-ELAN | 25 | SGD | 0.01 | 16 |
| YOLOv7 | E-ELAN | 55 | Adam | 0.01 | 8 |
| DETR | CNN + encoder-decoder transformer | 20 | AdamW | 1e-4 | 2 |
| EfficientDet | EfficientNet + BiFPN | 30 | AdamW | 5e-3 | 8 |
| RetinaNet | ResNet + FPN | 30 | Adam | 1e-5 | 8 |

*Fuente:* Tomado de (Abuhani et al., 2023).

En el estudio realizado por Deng et al. (2024), se llevó a cabo una evaluación de los modelos YOLOX-large (43.6 M de parámetros), YOLOv8 (54.2 M de parámetros) y DINO-SwinL-scale4 (217.2 M de parámetros) para detectar malezas capturadas en distintas estaciones. El dataset utilizado contiene 6,664 imágenes con 10,848 instancias para 8 clases de malezas divididas en 75%, 5% y 20% para el entrenamiento, validación y testeo. Este dataset se consiguió gracias a la unión del dataset CottonWeedDet12 del año 2021 y la otra parte fueron imágenes tomadas con cámaras portátiles en las granjas de investigación de la Universidad Estatal de Mississippi. Los tres modelos fueron entrenados por 40 épocas con un batch size de 8, se redimensionó las imágenes a 800 píxeles, para los modelos YOLO se usó el optimizador SGD con un learning rate de 0.01 mientras que para DINO se utilizó el optimizador AdamW con un learning rate de 0.0001. El framework usado fue PyTorch (versión 1.13.1) con Torchvision (versión 0.14.1) y CUDNN (versión 9.2) en una GPU NVIDIA RTX A6000. El mejor resultado fue de 0.987 mAP con el modelo DINO, mientras que YOLOX y YOLOv8 alcanzaron 0.974 y 0.966 de mAP,

respectivamente. Este rendimiento se puede explicar debido a que el modelo basado en Transformer (DINO) cuenta con cuatro veces más de número de parámetros que los modelos YOLO lo que también se traduce a más consumo de recursos de cómputo.

En el trabajo de Pavithra et al. (2023), se propone un sistema que usa un dron equipado con dos cámaras para detectar malezas y enfermedades de las plantas utilizando dos modelos YOLOv8. Los modelos fueron entrenados en un dataset de 3.713 imágenes de 50 especies de malezas con clases desbalanceadas. Se entrenó por 40 épocas con las imágenes redimensionadas a un tamaño de 416 x 416 píxeles aplicando técnicas de aumento de imágenes para reducir el impacto de las clases desbalanceadas. Los resultados obtenidos para la detección de malezas fueron de 0.86 en mAP@50 y 0.59 en mAP50-95, mientras que, en la detección de enfermedades se obtuvieron 0.46 mAP@50 y 0.30 mAP50-95. Estos resultados pueden deberse a que la tarea de detección de enfermedades fue más desafiante porque muestran patrones más irregulares, que a diferencia de la detección de malezas se obtuvieron mejores métricas debido a que las malezas por lo general comparten similitud entre sí.

En la investigación realizada por Iqbal et al. (2023), se explora diferentes modelos para detectar malezas y plantas de maíz. La adquisición del dataset se realizó en un campo de maíz que capturó imágenes a 640x480 y 1280x720 píxeles mediante dos Sensorbox. Las muestras fueron tomadas en un periodo de tres semanas para asegurar las diferentes etapas de crecimiento, al final se consiguió 3.574 imágenes de maíz y malezas. Se experimentó con los modelos Faster R-CNN, RetinaNet, FCOS, YOLO y DINO. Para los tres primeros modelos se utilizó un batch size de 32, un learning rate de 0.01 y el optimizador SGD. También se utilizó los modelos YOLOv5 medium y large con un batch size de 32 y 16. El modelo DINO usó un batch size de 4 y un learning rate de 0.0001. Todos los modelos utilizaron un tamaño de 800 píxeles y con los parámetros por defecto sin aplicar aumento de datos, además ocuparon 32 GB de GPU NVidia Tesla V100 con la versión de CUDA 11.1. Los resultados se resumen en la Tabla 6. donde el modelo DINO obtiene un mAP@50 de 75.8, inferior a los modelos YOLO con 9.6 puntos de diferencia, sin embargo, obtuvo mejores resultados que los modelos Faster R-CNN, RetinaNet y FCOS.

**Tabla 6**

*Resultados de rendimiento para la detección de malezas y maíz*

| Modelo | mAP@50 | mAP50:95 | AP50 maíz | AP50 malezas |
|---|---|---|---|---|
| Faster R-CNN FPN | 71.6 | 41.8 | 88.1 | 55.1 |
| RetinaNet R50 FPN | 68.0 | 40.2 | 89.7 | 46.3 |
| FCOS R50 FPN | 69.3 | 39.8 | 87.0 | 51.7 |
| YOLOv5 medium | **85.4** | 53.3 | 93.0 | **77.8** |
| YOLOv5 large | **85.4** | **53.9** | **93.7** | 77.0 |
| DINO Transformer | 75.8 | 45.0 | 92.3 | 59.3 |

*Fuente:* Tomado de (Iqbal et al., 2023).

Los autores Li & Shi (2024) usan el enfoque del aprendizaje semisupervisado para entrenar el modelo DETR para identificar brassica chinensis, una especie de col muy popular en el sur y sureste de China. El estudio se realizó en el invernadero de Shanghai Shixin Vegetable Cultivation Company ubicadas a 121º03' de latitud este y 37º07' de latitud norte de 80 metros de largo y 40 metros de ancho. Se utilizó 5 cámaras esféricas Hikvision's DS-2DC4223IW-D posicionadas a 2,5 metros de altura y un sistema UAV equipada con una cámara de acción Sony que graba a 1080p@60 FPS en cámara lenta. Se recolectó 439 imágenes a 1920x1024 píxeles que contienen 7 399 objetos y se repartió el 80% para el entrenamiento y el 20% para el testeo. El entrenamiento del modelo se realizó en 2 etapas, el calentamiento de datos etiquetados y el aprendizaje de datos semisupervisado, donde se aplicó técnicas de aumento de datos (recortes, volteos horizontales y transformaciones geométricas). Los hiperparámetros utilizados fueron el optimizador SGD con el momentum de 0.9, un batch size de 4 para los datos supervisados y 8 para los datos no supervisados, weight decay rate de 0.0001, learning rate de 0.0002 para el backbone, learning rate de 0.002 para el resto del modelo y un learning rate scheduler constante. Los resultados muestran que utilizando el 1% de los datos etiquetados del dataset de entrenamiento semisupervisado se alcanza un 70.0% de mAP y 89.3% de AP50 que es relativamente inferior al usar el 10% se obtiene 77.7 % de mAP y un 93.9% de AP50, pero se destaca por haber alcanzado resultados respetables con muy pocas muestras. Por otro lado, si se usa el enfoque completamente supervisado los

resultados son ligeramente superiores con 77.9 % de mAP y 93.8% de AP50. Una limitación presente es la dificultad del modelo para la implementación en escenarios de tiempo real.

En el estudio realizado por Borgogno et al. (2024), se desarrolló un sistema robusto para la detección multiclase de malezas en distintas etapas de crecimiento. La recolección de datos se realizó en la granja de Arganda del Rey en Madrid España para capturar cultivos de maíz con una superficie de 7400 $m^2$ y para la captura de cultivos de tomates se realizó en los campos comerciales de Santa Amalia ubicadas en Badajoz España en un espacio de 12000 y 14000 $m^2$. El equipo utilizado fue la cámara Sony ILCE-6300L de 24.2 megapíxeles montado en un UAV modelo Microdrones MD4-1000, las imágenes fueron capturadas a 11 metros de altura con una resolución de 6000x3376 píxeles en el mes de mayo de los años 2020 y 2021 aproximadamente a las 13:00 CET. En total se obtuvo 1133 imágenes de cultivos de maíz y 1845 imágenes de cultivos de tomate con 10 especies de malezas en general. El modelo utilizado para la detección fue DETR en combinación con los datasets generados con el modelo GAN, los hiperparámetros utilizados fueron el learning rate de 0.0004, el weight decay de 0.0004, batch size de 2 para el entrenamiento y 1 para la validación entrenados por un total de 20 épocas. El mejor resultado se obtuvo entrenando el dataset original más el aumento de datos mediante la red GAN x2 donde se alcanzó un mAP@50 de 0.513 y un mAP@75 de 0.171, independientemente de los resultados el modelo final tuvo un peso de 166.5 Mb en disco con 41.5 millones de parámetros, el tiempo de inferencia para el maíz es aproximadamente de 117.45 segundos por 2390 imágenes y para el tomate fue de 169.83 segundos sobre 3669 imágenes.

# CAPÍTULO II

# DESARROLLO

## 2.1. Planificación del proyecto

El presente trabajo de titulación busca desarrollar una aplicación web para la detección automática de malezas en campos de cultivo de maíz y/o papa mediante la red profunda Transformer. Con base en la fundamentación teórica descrita previamente, este capítulo se enfocará en aplicar las teorías y herramientas estudiadas para alcanzar los objetivos planteados. El desarrollo del proyecto está planificado para 6 meses de duración.

### 2.1.1. Metodología KDD

La metodología KDD se empleó para la elaboración del dataset y el entrenamiento/validación del modelo, cada una de las cinco fases se encuentran aplicadas a lo largo del presente capítulo. En la Tabla 7 se muestra una lista de actividades del presente trabajo adaptadas a cada una de las fases de la metodología, donde se señala la fase de la metodología, la actividad, el resultado y el tiempo estimado.

**Tabla 7**

*Planificación de las actividades del proyecto*

| Nro. | Fase de la metodología KDD | Actividad | Resultado por actividad | Tiempo estimado |
|---|---|---|---|---|
| 1 | Selección | Plantar malezas faltantes del dataset de papas. | Campo con las malezas faltantes. | 6 semanas |
| | | Capturar las malezas con un dron. | Imágenes/Videos de alta resolución de las malezas. | 4 semanas |
| | | Separar las imágenes inválidas del dataset. | Dataset libre de imágenes de baja | 1 semana |

| | | | | |
|---|---|---|---|---|
| | | | resolución, baja altura o borrosas. | |
| 2 | Preprocesamiento | Recortar las imágenes válidas manteniendo una relación de aspecto 1:1. | Imágenes preparadas para el proceso de anotación. | 2 semanas |
| 3 | Transformación | Subida y anotación de las imágenes en la plataforma Roboflow. | Malezas y cultivos de cada imagen anotadas por un cuadro delimitador. | 7 semanas |
| | | Separación del dataset en conjuntos de entrenamiento, validación y testing. | Dataset separado en 3 carpetas con sus respectivas anotaciones. | 1 semana |
| 4 | Minería de datos | Preparación del entorno de entrenamiento. | Librerías y modelo Transformer listo para entrenar. | 1 semana |
| | | Entrenamiento del modelo Transformer. | Modelo entrenado serializado. | 1 semana |
| | | Ajustar los hiperparámetros del modelo. | Conseguir el mejor modelo para la tarea de detección de malezas. | 2 semanas |
| | | Recolectar resultados de rendimiento con Wandb. | Gráficas y medidas de rendimiento del modelo por cada entrenamiento. | 1 semana |
| 5 | Interpretación/ Evaluación | Realizar tablas estadísticas de los resultados obtenidos. | Análisis de los resultados de entrenamiento. | 1 semana |
| | | Selección del mejor modelo. | Modelo serializado listo para producción. | 1 semana |
| | | Inferencia de las malezas. | Prueba del modelo con el conjunto de datos de testing. | 2 semanas |

*Fuente:* Autoría propia.

### 2.1.2. Instrumentos

Para la captura de imágenes de las malezas se utilizará el dron DJI Mavic 2 Pro, en la Tabla 8 se especifican sus características técnicas.

**Tabla 8**

*Características del dron DJI Mavic 2 Pro*

| Especificación | Detalles |
|---|---|
| Sensor | 1’’ CMOS<br>20 millones de pixeles |
| Lente | Campo de visión de 77°<br>Apertura f/2,8-f/11 |
| Tamaño de imagen | 5472×3648 |
| Resolución de video | 4K: 3840×2160<br>2,7K: 2688x1512<br>FHD: 1920×1080 |
| Velocidad máxima (sin viento) | 72 km/h |
| Altitud máxima de despegue | 6000 metros |
| Tiempo máximo de vuelo (sin viento) | 31 minutos (velocidad constante de 25 km/h) |
| Resistencia máxima de la velocidad del viento | 29-38 km/h |
| Capacidad de la batería | 3850mAh |
| Almacenamiento | 8 GB |

*Fuente:* Adaptado de (DJI, 2024).

El entrenamiento del modelo se realizó en la plataforma colaborativa “Kaggle” debido a que ofrece un entorno generoso de recursos de hardware para el aprendizaje automático. Además, permite a los usuarios participar en competiciones, acceder a cientos de datasets y notebooks de la comunidad. En la Tabla 9 se puede observar los recursos de hardware ofrecidos por Kaggle.

**Tabla 9**

*Recursos de Kaggle*

| Recurso | Detalles |
|---|---|
| CPU | 29 GB de RAM<br>12 horas por sesión |
| Almacenamiento | 73.1 GB por sesión |
| GPUs NVIDIA TESLA T4 x2, P100 | 15 GB de memoria por GPU<br>30 horas de uso de 12 horas por sesión |
| TPU (Tensor Processing Units) VM v3-8 | 20 horas de uso de 9 horas por sesión |
| Almacenamiento privado para modelos | 107.37 GB |
| Almacenamiento privado para datasets | 107.37 GB |

*Fuente:* Adaptado de (Kaggle, 2024).

## 2.2. Preparación del dataset

El entrenamiento de un algoritmo de aprendizaje profundo requiere de cuatro ingredientes principales: un dataset, un modelo/arquitectura, una función de pérdida y un método de optimización. El dataset permite tener un enfoque claro del problema a solucionar (Rosebrock, 2019).

El enfoque tomado para el presente trabajo fue la detección de objetos debido a la naturaleza del problema de cuantificación de malezas, para ello se elaboró un dataset con imágenes de campos de cultivo de papa y malezas. Como se muestra en la Tabla 10, se consideró 4 tipos de plantas que son la papa, el diente de león, la lengua de vaca y el kikuyo. Además, se optó por agregar la clase “Otros” para incluir el resto especies de plantas que no fueron mencionadas anteriormente.

**Tabla 10**

*Plantas que conforman el dataset*

| Planta | Ilustración |
|---|---|
| Papa (Solanum tuberosum) |  |
| Diente de león (Taraxacum officinale) |  |
| Lengua de vaca (Dracaena trifasciata) |  |
| Kikuyo (Pennisetum clandestinum) |  |
| Otros |  |

*Fuente:* Autoría propia.

Para crear el dataset completo se utilizó las imágenes de los trabajos de titulación previos junto con las imágenes capturadas por autoría propia.

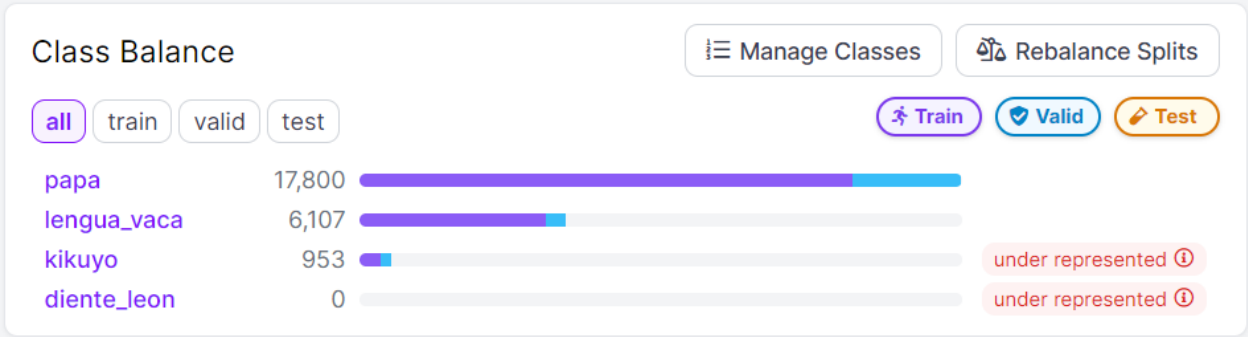
Las imágenes de los trabajos titulación previos fueron tomadas en 7 terrenos de la zona de Cuesaca, San Gabriel y el Ángel de la provincia del Carchi (Vinueza, 2024), a partir de este

conjunto de datos se seleccionó 146 imágenes limpias sin ningún tipo de preprocesamiento o anotación.

Como se puede observar en la Figura 10, se logró obtener 17,800 instancias de papas, 6,107 instancias de lengua de vaca, 953 instancias de kikuyo y 0 ejemplares de diente de león. Las clases que tuvieron problemas de baja representación fueron el kikuyo y el diente de león. Este fue un problema muy importante que se observó, puesto que un dataset con insuficientes datos puede traer problemas de desajuste en el modelo.

**Figura 10**

*Número de clases obtenidas en el dataset incompleto*




*Nota.* La figura muestra el número de plantas anotadas en el dataset obtenido solo de los trabajos de titulación previas. Autoría propia.

### 2.2.1. Cultivo de malezas

Debido a la pobre representación de malezas, se optó por cultivar las malezas faltantes en un terreno propio que se encuentra ubicado en la comunidad de Tunibamba del cantón Cotacachi.

El primer paso fue la recolección de malezas de diente de león y kikuyo, este tipo de malezas normalmente se encuentran en campos o en vías públicas. En la Figura 11 se observa un ejemplar de diente de león en su etapa de crecimiento. Se consideró plantas semejantes para simular la población de diente de león en un cultivo de papas. En el caso del kikuyo, se recolectó las plantas que tuvieran de 15 a 20 cm de tallo, estas medidas son aproximadas del tamaño del diente de león en los campos de papa.

**Figura 11**

*Diente león*

*Fuente*: Autoría propia.

Para sembrar las malezas en el terreno, primero se echó una herbicida para eliminar las diversas malezas que presentes. Luego de dos semanas se preparó el terreno arando con un tractor. Los surcos fueron realizados a la medida usada para sembrar maíz, es decir, con una distancia aproximada de 60 cm entre surcos. Las malezas fueron trasplantadas a una distancia aproximada de 30 cm entre sí, como se observa en la Figura 12.

**Figura 12**

*Siembra del diente de león*

*Fuente*: Autoría propia.

Una vez sembrada las malezas, se los cuidó regándolos en los días soleados y removiendo las malezas no deseadas que aparecieron naturalmente. Todo el proceso de cultivo de las malezas

tomó alrededor de 6 semanas, donde la mayoría del tiempo fue invertido en recolectar y cultivar gradualmente las malezas mientras se los recolectaba, la parte restante del tiempo fue utilizado para dar cuidados a la maleza y evitar que se sequen.

Como se observa en la Figura 13, se realizó el deshierbe a las malezas que surgieron naturalmente y se regó las malezas trasplantadas.

**Figura 13**

*Cuidados realizados al campo de malezas*


*Nota*: a) Riego del campo de malezas, b) deshierbe de las malezas no trasplantadas y c) ejemplares de las malezas no trasplantadas. Autoría propia.

### 2.2.2. Captura de malezas

Una vez que se completó el cultivo de las malezas, se organizó las visitas al campo para realizar las respectivas capturas con el dron DJI Mavic Pro-2. En la mayoría de los trabajos, optan por una altura de 5 o 10 metros, esto tiene una relación directa con la cantidad de área que se cubre, puesto que, a mayor altura se puede detectar más cantidad de malezas en menos tiempo en comparación de una altura baja, se emplea más tiempo para cubrir la misma cantidad de área (Vinueza, 2024).

En la toma de datos, se procuró realizarlo entre 9 y 10 metros de altura con una velocidad de vuelo de 1 m/s con la finalidad de mantener los mismos parámetros de las imágenes del dataset de los trabajos de titulación previos.

Las malezas fueron capturadas en formato de video con una resolución de 3840x2160 píxeles a 30 fotogramas por segundo. Los drones de la marca DJI cuentan con un software llamado DroneDeploy para programar la altura, el recorrido y la velocidad de vuelo; sin embargo, no se empleó este software debido a que el terreno que se utilizó tiene una pendiente considerable que puede afectar los parámetros programados, en su lugar, se capturó los videos manualmente. En la Tabla 11 se resume las visitas realizadas al campo de malezas.

**Tabla 11**

*Resumen de visitas al campo de malezas*

| **Ubicación** | **Fecha de visita** | **Etapa de crecimiento** | **Videos capturados y duración total** |
|---|---|---|---|
| Campo de malezas ubicado en Cotacachi, comunidad Tunibamba de Bellavista, 0°19'00.1"N 78°15'56.0"W. | 11/05/2024, 09:56 | 15 – 20 días aprox. | 5 videos, duración total 15,55 min. |
| | 17/05/2024, 09:39 | 20 – 25 días aprox. | 7 videos, duración total 28,00 min. |

*Fuente:* Autoría propia.

En total se recolectó 1,369 frames de todos los videos, sin aplicar ningún tipo de selección o preprocesamiento de imagen, la Figura 14 muestra un ejemplar de las malezas capturadas con el dron DJI Mavic Pro-2.

**Figura 14**

*Ejemplar del campo de malezas*

*Nota.* Entre las malezas que se capturaron son el diente de león, el kikuyo y la lengua de vaca, las cuales fueron capturadas a una altura de 9 - 10 metros por el dron DJI Mavic 2 Pro. Autoría propia.

### 2.2.3. Selección y preprocesamiento de imágenes

La selección, preprocesamiento y anotación de imágenes son las principales fases para la elaboración de un dataset de calidad, así mismo, son las fases que más tiempo y esfuerzo requieren. En el proceso de selección de imágenes se descartó con considerando lo siguiente:

- **a.** Imágenes tomadas en las zonas con pendientes altas.
- **b.** Imágenes con ruido y borrosas.
- **c.** Imágenes capturadas al principio y al final del video.
- **d.** Imágenes capturadas cuando el dron se quedaba estático durante la grabación.
- **e.** Imágenes con plantas difíciles de reconocer.

En las zonas planas del terreno hubo fotografías con malezas que no se podían identificar a simple vista. Este problema se presentó principalmente con las imágenes tomadas en la primera

visita, específicamente con la planta de diente de león. En cambio, con las otras plantas como la lengua de vaca, kikuyo o papa, no hubo problema en distinguirlas.

En la fase de preprocesamiento se realizó el recorte de las imágenes elegidas a 1920x1920 píxeles del centro de la fotografía como se observa en la Figura 15. Una de las razones por las que se optó por recortar las imágenes a un formato cuadrado es para aprovechar los beneficios y optimizaciones que tienen las librerías de aprendizaje profundo para realizar cálculos matriciales (Rosebrock, 2019). Las dimensiones que se eligen normalmente para entrenar un modelo son números múltiplos de 32, es por ello que se eligió 1920 píxeles; otro motivo para elegir un valor alto fue para experimentar en diferentes dimensiones mediante la redimensión por código y hallar un valor óptimo. Un beneficio indirecto que se determinó fue la reducción del peso en megabytes (Mb) de cada imagen, donde se logró reducir el peso original de 3 Mb y 4 Mb a 1.5 Mb y 2.5 Mb, dando como resultado un peso final que oscilan los 1.06 Gb para el dataset completo.

**Figura 15**

*Muestra de las imágenes seleccionadas y recortadas a 1920x1920 píxeles*

*Nota:* Estas imágenes cumplen con los criterios de selección previamente mencionados. Autoría propia.

Para realizar estas tareas, se escribió un script en python que clasifica las imágenes en una carpeta de acuerdo al input del usuario, luego recorta las imágenes manteniendo la relación de aspecto en un tamaño que especifica el usuario. Las librerías base usadas en el script fueron OpenCV y imutils.

Una vez finalizado todo el proceso selección y preprocesamiento, se obtuvo un conjunto de 400 imágenes con las distintas plantas que se menciona en la Tabla 10.

### 2.2.4. Anotación de imágenes

La anotación de las imágenes es el siguiente paso, previo al entrenamiento del modelo de aprendizaje profundo, para ello se consideró el uso de la plataforma Roboflow que permite etiquetar objetos mediante un cuadro delimitador. Como se observa en la Figura 16, la herramienta de anotación cuenta con 2 paneles laterales que permiten el etiquetado a manos libres, dejar comentarios, deshacer anotaciones, distinción de clases por colores, etc.

**Figura 16**

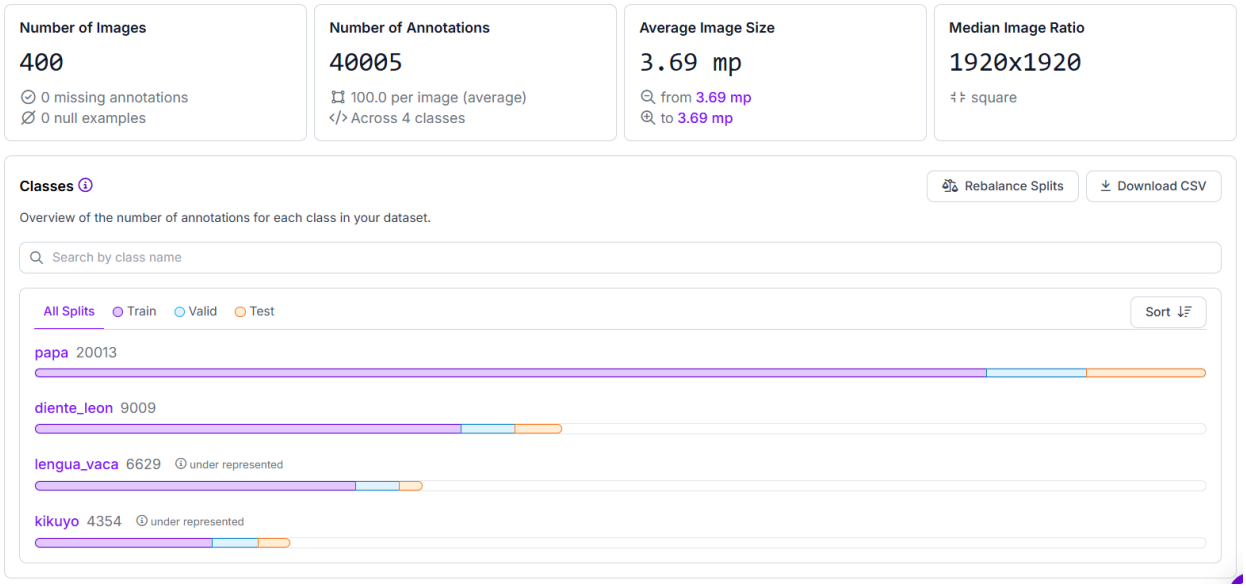
*Pantalla de anotación de Roboflow*


*Fuente*: Autoría propia.

El proceso de anotación se llevó a cabo por lotes de 10 a 50 imágenes, donde se procuró anotar todas las plantas que aparecía en cada una de ellas. En la primera versión del dataset solo se consideró 4 clases de plantas como papa, diente de león, lengua de vaca y kikuyo, debido a que las imágenes capturadas en la primera visita tenían casi nula presencia otro tipo de plantas. En la Figura 17 se detallan las anotaciones obtenidas en la primera versión del dataset.

**Figura 17**

*Detalles de la primera versión del dataset*




*Fuente:* Autoría propia.

Sin embargo, el factor decisivo para incluir la clase “otros” fue gracias las imágenes recolectadas de la segunda visita al campo de malezas, donde se encontró varias plantas aleatorias que no se consideró al principio como una clase individual.

El criterio para etiquetar a una planta con su respectiva clase fue que se pueda identificar la planta del resto con o sin realizar zoom; en cambio, las plantas que, a pesar de realizar zoom, se mostraban como una mancha o pertenecía a otra especie, se lo etiquetó en la clase “otros”. Por otra parte, gracias a las condiciones climáticas favorables a la hora de la captura de malezas la mayoría de las imágenes recolectadas mantienen una iluminación, contraste y saturación uniforme.

### 2.2.5. Exportación del dataset

Una vez finalizado el proceso de anotación de las plantas se exportó el dataset en un archivo comprimido con extensión .zip. Para realizar este proceso, Roboflow ofrece la opción de generar versiones, donde el usuario pasa por cinco etapas para generar una versión del dataset que se detalla en la Tabla 12.

**Tabla 12**

*Preprocesamiento de Roboflow*

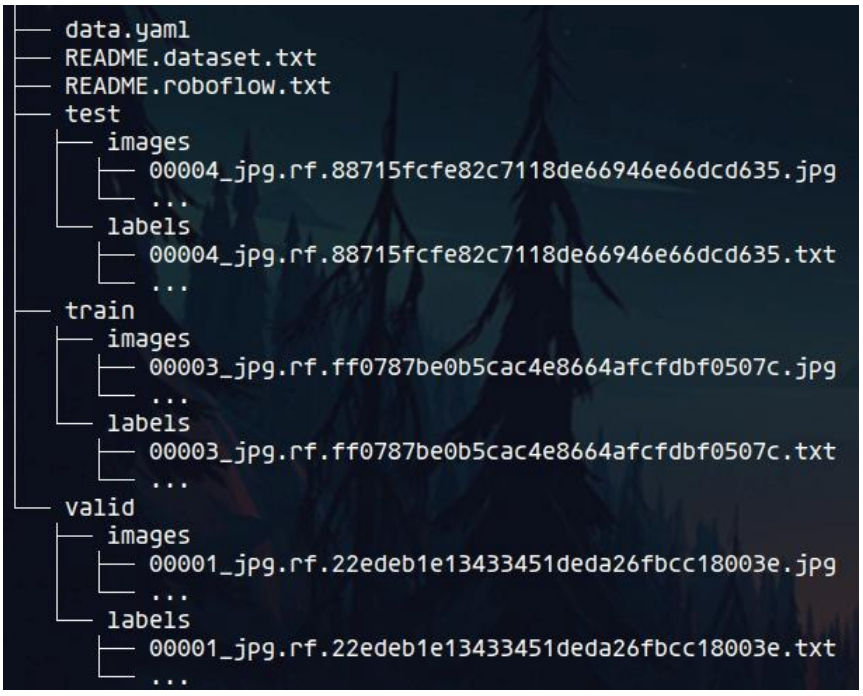
| Etapa | Descripción | Valor |
|---|---|---|
| 1. Seleccionar la fuente de imágenes | Se elige las imágenes anotadas para incluir en el dataset. | 400 imágenes anotadas |
| 2. División del dataset | Dividir el dataset final en conjuntos de entrenamiento, validación y testeo. | Entrenamiento: 80%, 320 imágenes.<br>Validación: 10%, 40 imágenes.<br>Testeo: 10%, 40 imágenes. |
| 3. Preprocesamiento | Aumentar el rendimiento del modelo aplicando transformaciones a todas las imágenes del dataset. | Ninguna |
| 4. Aumento | Crear nuevas imágenes para el entrenamiento del modelo y aprenda a generalizar a partir más variantes de los objetos. | Ninguna |
| 5. Creación | Finaliza con una revisión rápida de los preprocesamientos aplicados. | Ninguna |

*Fuente:* Autoría propia.

Una vez creada la versión del dataset en Roboflow, el proceso de exportación es sencillo. Primero se eligió el formato de anotaciones de dataset, este formato depende fuertemente de la librería y el modelo a entrenar, para este caso de estudió se seleccionó el formato de anotación YOLOv8. En la Figura 18 se presenta estructura de directorios generados.

**Figura 18**

*Estructura de directorios YOLOv8*




*Fuente:* Autoría propia.

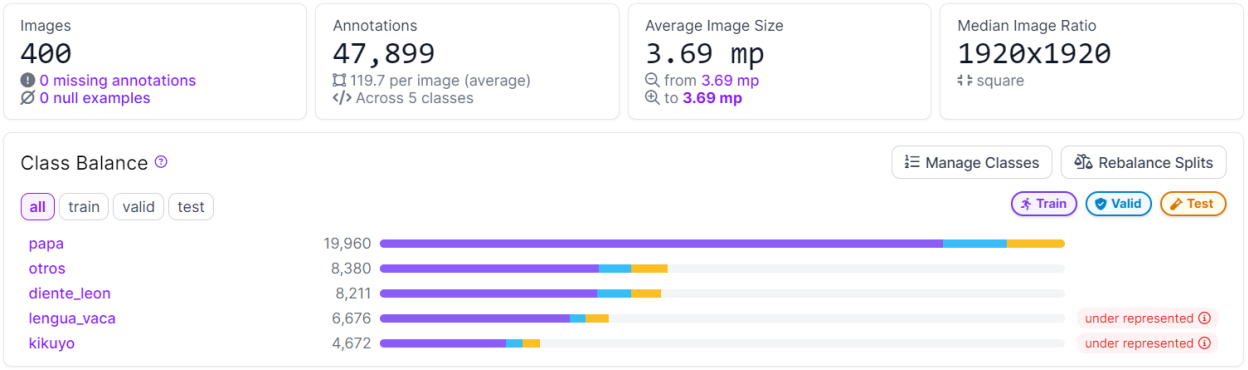
A continuación, se describe cada uno de los ficheros y directorios generados:

- **data.yaml**: Este fichero es un formato de configuración que permite definir la raíz del proyecto, las rutas relativas para el conjunto de entrenamiento, validación y prueba y un diccionario de los nombres de las clases.
- **Directorios de entrenamiento, validación y pruebas:** Estos directorios son utilizados para agrupar las imágenes para cada conjunto, en cada directorio hay el subdirectorio “images” que es donde residen las imágenes y el subdirectorio “labels” es donde residen las anotaciones de cada imagen.
- **Ficheros README:** Adicionalmente Roboflow incluye un par de ficheros “README” para incluir metadatos del dataset.

El formato de anotación de YOLOv8 se caracteriza por utilizar un archivo de texto plano por cada imagen para almacenar las etiquetas, estas deben seguir la siguiente estructura para considerarse válidas: class x_center y_center width height (Jocher, Rizwan, et al., 2024).

**Figura 19**

*Segunda versión del dataset de malezas*



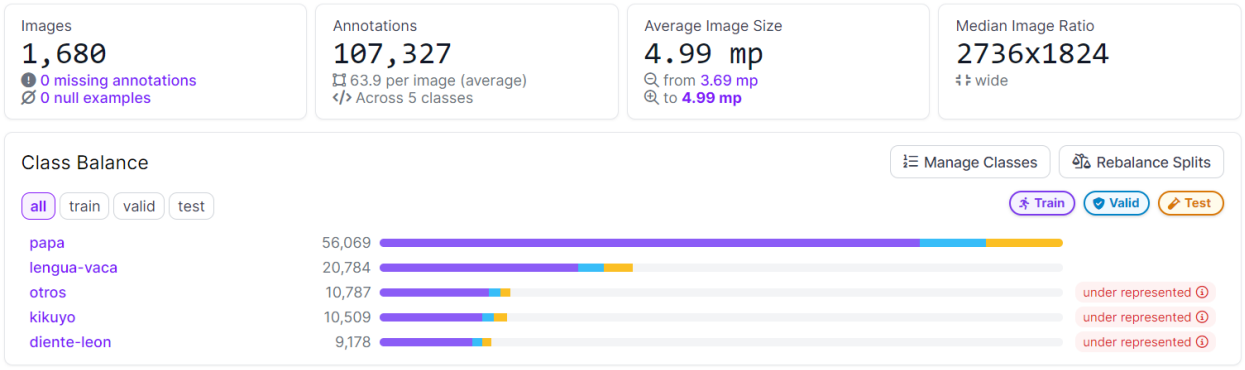

*Fuente:* Autoría propia.

De acuerdo con la Figura 19, se obtuvo en total 400 imágenes de 1920x1920 píxeles con un total de 47,899 anotaciones. Cada una de las clases tiene como mínimo 4,672 etiquetas de las cuales el 80% se pertenece a conjunto de entrenamiento, el 10% pertenece al conjunto de validación y el 10% restante pertenece al conjunto de testeo. Inevitablemente, la clase papa es la única que tiene demasiada representación frente al resto de clases que se encuentran moderadamente iguales.

La tercera versión del dataset se creó gracias a la colaboración de otro dataset anotado de malezas, como se observa la Figura 20, esta nueva versión tiene 1,680 ejemplares, con 107,327 anotaciones y un tamaño promedio de 2736x1824 píxeles. Esta versión se obtuvo simplemente por la mezcla de ambos datasets sin comprobar la calidad de las imágenes, etiquetas y tamaño.

**Figura 20**

*Tercera versión del dataset de malezas*

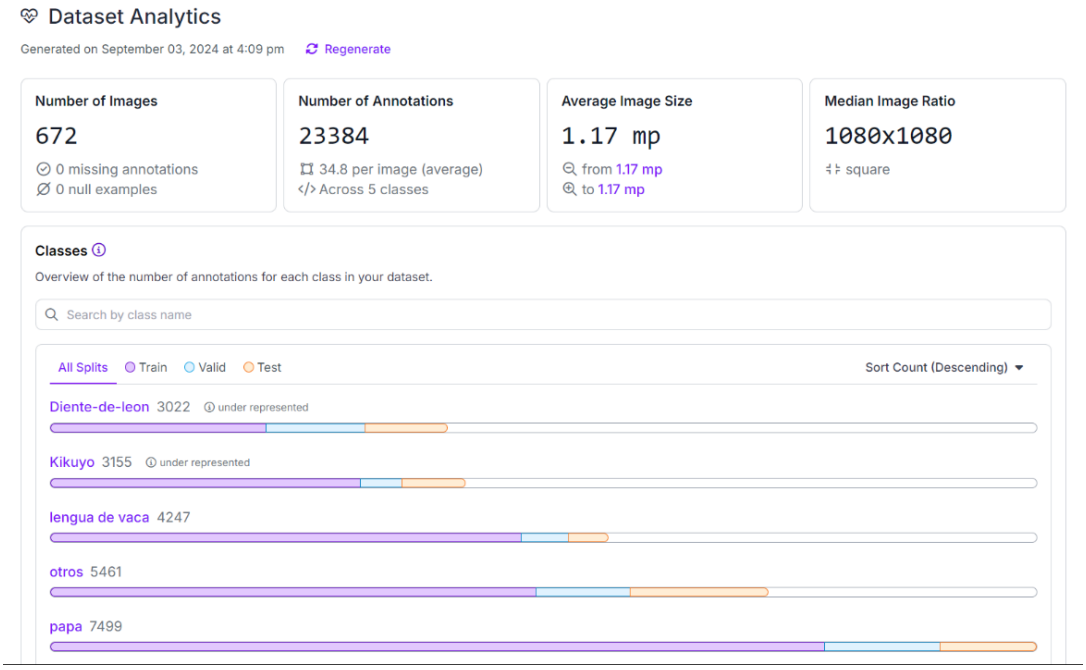



*Fuente:* Autoría propia.

Con el afán de mejorar el dataset, se realizó varios ajustes a la versión 3 del dataset; como se observa en la Figura 21 en la cuarta versión se redujo el número de imágenes a 672 recortadas a 1080x1080 píxeles, también el número de anotaciones de la clase papa se redujo a 7,499 lo que indirectamente afectó a las otras clases puesto que varias de ellas compartían la misma ubicación en el campo.

**Figura 21**

*Cuarta versión del dataset*



*Fuente:* Autoría propia.

En la Tabla 13, se resume las 4 versiones del dataset creados, con las que se experimentaron para obtener las mejores métricas de rendimiento del modelo. Cabe destacar que durante las experimentaciones se descartaron el uso de varios datasets debido a que tenían problemas de balanceo de clases.

**Tabla 13**

*Resumen de los datasets obtenidos*

| Nro. | Detalles | Nro. imágenes | Papas | Lengua de vaca | Kikuyo | Diente de león | Otros | Total de etiquetas |
|---|---|---|---|---|---|---|---|---|
| 1 | Dataset sin la clase “otros”. | 400 | 20 013 | 6 629 | 4 354 | 9 009 | 0 | 24 860 |
| 2 | Dataset creado con las plantas cultivadas. | 400 | 19 960 | 6 676 | 4 672 | 8 211 | 8 380 | 47 899 |
| 3 | Dataset creado a partir de la unión de dos datasets. | 1 680 | 56 069 | 20 784 | 10 509 | 9 178 | 10 787 | 107 327 |
| 4 | Dataset ajustado las etiquetas y el tamaño. | 672 | 7 499 | 4 247 | 3 155 | 3 022 | 5 461 | 23 384 |

*Fuente:* Autoría propia.

## 2.3. Entrenamiento del modelo Transformer

El entrenamiento del modelo Transformer corresponde a la fase de minería de datos de la metodología KDD. El objetivo de esta fase es entrenar el modelo hasta alcanzar el mejor rendimiento de acuerdo con las métricas de detección de objetos, mediante la optimización de los

hiperparámetros del modelo. Al finalizar esta fase se contará con todos los resultados, modelos y gráficas que permitirá evaluar y validar el mejor modelo alcanzado.

### 2.3.1. Selección del modelo Transformer

La definición del modelo se realizó investigando los diferentes modelos Transformers preparados para la detección de objetos existentes en la comunidad. Uno de los primeros modelos encontrados fue DETR presentado el autor (Carion et al., 2020), este fue el primer modelo basado en Transformers para resolver problemas de detección y segmentación de objetos; uno de sus principales atractivos fue que el modelo eliminaba el uso de NMS, que a diferencia de los detectores como YOLO lo utilizan durante posprocesamiento de la imagen para realizar la inferencia de objetos.

A partir de DETR surgieron modelos como Deformable-DETR para acelerar la convergencia de entrenamiento, el modelo Efficient DETR para reducir los costos computacionales utilizados, el modelo Lite DETR para mejorar la eficiencia de codificador, DINO mejorar la inicialización de consultas (Zhao et al., 2024).

A pesar de las mejoras introducidas, ningún modelo Transformer alcanzaba el rendimiento en términos de velocidad de inferencia a los modelos de vanguardia de la familia YOLO. No obstante, en 2023 se lanzó un nuevo modelo llamado RT-DETR propuesto por el autor (Zhao et al., 2024), la arquitectura implementada mejoraba drásticamente la velocidad de inferencia manteniendo el mismo rendimiento o superior, de esta forma se alineó al nivel de los famosos modelos YOLO y es considerado como el sucesor directo del modelo DETR.

Para el desarrollo del presente trabajo se eligió el modelo RT-DETR en su variante large (con 32 millones de parámetros) debido a las capacidades demostradas en su artículo.

### 2.3.2. Preparación del entorno del entrenamiento

Como se definió previamente, el entorno para realizar las pruebas fue Kaggle. En esta plataforma se cargó la primera, segunda, tercera y cuarta versión del dataset de malezas en formato YOLOv8, luego se creó un jupyter notebook para entrenar el modelo. La versiones del lenguaje y librerías usadas fueron: Python 3.10.14, CUDA 12.6, PyTorch 2.4.0, ultralytics 8.3.49, coco-eval 0.0.4 y pycocotools 2.0.8.

Primero se instaló las librerías necesarias para entrenar el modelo que son coco-eval, pycocotools, ultralytics y wandb. Después, se inicializa la plataforma de wandb para registrar todas las métricas del proceso de entrenamiento y validación. La decisión de utilizar específicamente esta plataforma para registrar las métricas de entrenamiento en lugar de Tensorboard fue porque wandb ofrece un servicio gratuito en la nube para guardar las métricas y facilita bastante el proceso de análisis y comparaciones de las métricas de rendimiento, puesto que permite analizar resultados de varios entrenamientos en conjunto. A diferencia de wandb, guardar los resultados de rendimiento en la nube con Tensorboard resulta más complicado.

Luego, se definió un script para crear un fichero de configuración "data.yaml" admitido por la librería de ultralytics con los parámetros personalizados para entrenar el modelo. Dentro de este archivo se define parámetros como la ruta de dataset y los subdirectorios de entrenamiento, testeo y validación.

Una vez definido el fichero de configuración, se utiliza el modelo RT-DETR con los parámetros personalizados para las tareas de entrenamiento (train), afinamiento (tune) y validación (validación), así mismo las métricas generadas en estos procesos son almacenadas en la plataforma de wandb. Finalmente, el modelo resultante se guarda en disco junto a las gráficas de rendimiento del modelo.

### 2.3.3. Definición de los hiperparámetros de entrenamiento y aumento de datos

Para el entrenamiento del modelo, primero se fijó los hiperparámetros que se va a utilizar y que posteriormente se van a ajustar. En la Tabla 14 se detallan todos los hiperparámetros utilizados.

**Tabla 14**

*Hiperparámetros del modelo baseline*

| **Hiperparámetro** | **Valor** |
|---|---|
| Épocas (epochs) | 100 |
| Tamaño de imagen (imgsz) | 640 |
| Valor de parada temprana (patience) | 20 |

| | |
|---|---|
| Tamaño de lote (batch) | 8 |
| Optimizador (optimizer) | AdamW |
| Programador de tasa de aprendizaje coseno (cos_lr) | False |
| Abandono aleatorio (dropout) | 0.0 |
| Tasa de aprendizaje (lr0) | 0.001111 |
| Desintegración de pesos (weight_decay) | 0.0005 |
| Beta 1 (momentum) | 0.9 |
| Épocas de calentamiento (warmup_epochs) | 3.0 |
| Beta 1 inicial para la fase de calentamiento (warmup_momentum) | 0.8 |
| Tasa de aprendizaje inicial para la fase de calentamiento (warmup_bias_lr) | 0.0 |

*Nota:* El nombre escrito entre paréntesis corresponde al nombre del parámetro escrito explícitamente requerido por la librería ultralytics. Adaptado de (Jocher, Knobloch, et al., 2024).

A parte de la definición de los hiperparámetros, también se utilizó la técnica de aumento de datos. Se optó por utilizar esta técnica para mejorar representación de nuestras imágenes y ayudar a al modelo a generalizar mejor, es decir, que pueda identificar los objetos en diferentes condiciones.

Según Gallagher (2023), los modelos de inteligencia artificial aprenden mejor cuando son entrenados con datos únicos y variados, muchas veces se puede mejorar el rendimiento del modelo agregando imágenes aumentadas al conjunto de datos original.

En la Tabla 15, se listan cada uno de los hiperparámetros de aumento utilizados. En total se emplearon nueve tipos de aumento, es decir, cualquier dataset que se utilice para el entrenamiento, esta aumentará nueve veces el número de imágenes.

**Tabla 15**

*Hiperparámetros para el aumento de datos*

| Parámetro | Rango | Valor |
|---|---|---|
| Aumento de matiz HSV | 0.0 – 1.0 (fracción) | 0.015 |
| Aumento de saturación HSV | 0.0 – 1.0 (fracción) | 0.7 |
| Aumento del brillo HSV | 0.0 – 1.0 (fracción) | 0.4 |
| Traslado de imagen horizontal/vertical | 0.0 – 1.0 (fracción) | 0.1 |
| Escalado de imagen | >= 0.0 (ganancia) | 0.5 |
| Volteo de imagen izquierda a derecha | 0.0 – 1.0 (probabilidad) | 0.5 |
| Mosaico de imágenes | 0.0 – 1.0 (probabilidad) | 1.0 |
| Borrado aleatorio | 0.0 – 0.9 (probabilidad) | 0.4 |
| Recorte fraccionario para clasificación | 0.0 – 1.0 (fracción) | 1.0 |

*Fuente:* Adaptado de (Jocher, Knobloch, et al., 2024).

### 2.3.4. Entrenamiento del modelo baseline

El objetivo de entrenar el modelo en este punto fue conseguir las métricas baseline para tener como punto referencia para las modificaciones que se realicen posteriormente. Estas métricas se consiguieron sin alterar ningún hiperparámetro, como especifican en la Tabla 14 y Tabla 15.

Los únicos parámetros alterados fueron la parada temprana (patience) y tamaño de lote (batch). El primero originalmente tenía el valor de 100, sin embargo, este valor se redujo a 20 debido a que las épocas de entrenamiento totales fueron 100 y consecuentemente no iba surtir efecto. El segundo parámetro alterado fue el tamaño de lote puesto que este depende directamente con la capacidad de GPU disponible en el entorno de entrenamiento, se redujo el tamaño de lote 16 a 8 porque se dio problemas desbordamiento de memoria en las 30 GB de GPU ofrecidas por Kaggle.

### 2.3.5. Experimentación con diferentes optimizadores

El propósito de experimentar con diferentes optimizadores fue para analizar el comportamiento de cada uno de ellos en la tarea de detección de malezas. La importancia de realizar esta fase recae en que cada optimizador es un algoritmo diferente y basado en la naturaleza de cada dataset puede mostrar mejores o peores resultados.

Los optimizadores usados en esta fase fueron Adam, RAdam, NAdam y SGD. El optimizador AdamW no se consideró puesto que este ya se utilizó para entrenar el modelo baseline. La configuración de los hiperparámetros se mantuvo las mismas que se utilizaron en el entrenamiento del modelo baseline.

Una vez finalizado la experimentación se descartaron varios optimizadores basados en el puntaje alcanzado en las métricas de validación del modelo.

### 2.3.6. Experimentación con el modo automatizado de optimización de hiperparámetros

El modo de optimización de hiperparámetros automático se basa en utilizar algoritmos genéticos para mutar el conjunto de hiperparámetros existentes en uno nuevo hasta que el modelo alcance un rendimiento bastante optimizado, este modo explora el espacio de hiperparámetros disponibles de manera eficiente y evita los mínimos locales (Jocher, Noyce, et al., 2024). El motivo para experimentar con el modo de optimización automática fue para conocer su efectividad y viabilidad para el problema de detección de malezas.

Este modo se encuentra integrado en la librería ultralytics a través del método “tune” y requiere un parámetro adicional que es iteraciones (iterations), este parámetro permite entrenar el modelo las veces que se especifique, no se debe confundir con el parámetro de épocas.

Durante cada iteración los hiperparámetros se actualizan y debido a que el proceso de mutación implica aleatoriedad, se recomienda ejecutar el proceso completo de iteraciones una sola vez para mantener la coherencia y variabilidad (Jocher, 2024).

### 2.3.7. Ajuste de hiperparámetros

El ajuste de los hiperparámetros fue una de las fases más importantes para conseguir el mejor rendimiento del modelo. En este punto se experimentó modificando los hiperparámetros

fijados anteriormente. El ajuste de hiperparámetros principalmente se basó en probar diferentes tasas de aprendizaje (learning rate) sobre los optimizadores Adam, AdamW y RAdam.

En la Tabla 16, se detallan los hiperparámetros cambiados para entrenar por cada optimizador sin considerar la tasa de aprendizaje.

**Tabla 16**

*Ajuste de hiperparámetros sin considerar la tasa de aprendizaje y el optimizador*

| Hiperparámetro | Valor | Descripción |
|---|---|---|
| Programador de tasa de aprendizaje coseno (cos_lr) | True | Ayuda a mejorar la convergencia utilizando la curva del coseno. |
| Abandono aleatorio (dropout) | 0.5 | Ayuda a evitar el sobreajuste apagando las redes conectadas de manera aleatoria. |
| Épocas de calentamiento (warmup_epochs) | 5.0 | Ayuda a estabilizar el entrenamiento. |

*Fuente:* Adaptado de (Jocher, Knobloch, et al., 2024).

#### 2.3.7.1. Ajuste manual de la tasa de aprendizaje

El ajuste manual se realizó partiendo desde la tasa de aprendizaje común que se utiliza para los optimizadores de la familia Adam es decir 0.001. Después de fijar el valor base, se eligió una tasa de aprendizaje ligeramente superior de 0.005, también se consideró utilizar una tasa de aprendizaje considerablemente baja a 0.001, en este caso se utilizó la tasa de aprendizaje de 0.0001 junto a una desintegración de pesos (weight decay) de 0.0001, estos valores son utilizados en el artículo (Zhao et al., 2024) para entrenar el modelo RT-DETR en el famoso dataset COCO.

#### 2.3.7.2. Ajuste de la tasa de aprendizaje adaptativo

La tasa de aprendizaje adaptativo es una fórmula que se basa en el número de clases para generar una tasa de aprendizaje ajustada, funciona muy bien con tamaños de dataset pequeños (Jocher, 2023). Esta fórmula es la que emplea la librería ultralytics en su función “build optimizer” cuando se utiliza el modo “auto” y se visualiza en la ecuación 2.

$$learning\ rate = round\left(\frac{0.002 \times 5}{4 + number\ of\ classes}, 6\right) \quad (2)$$

*Fuente:* Tomado de (Jocher, 2023).

Como se observa en la Tabla 13, la mayoría de las versiones de los dataset empleados en este trabajo tienen como alrededor 672 imágenes y resultó conveniente emplear esta fórmula para su experimentación. Primero, se utilizó la tasa de aprendizaje generada para las cinco clases correspondientes de nuestro dataset, luego se experimentó fijando el número de clases a cero para obtener una tasa de aprendizaje alto, y finalmente se utilizó el número de clases 10 para obtener una tasa de aprendizaje bajo.

### 2.3.8. Experimentación con imágenes a diferentes escalas

En el entrenamiento de un modelo de inteligencia artificial para problemas de visión por computadora, el tamaño de la imagen influye directamente en el tiempo empleado para el entrenamiento y en el rendimiento del modelo.

Inicialmente se fijó el parámetro del tamaño de imagen a 640x640 píxeles, una vez finalizado el ajuste de hiperparámetros, el siguiente paso fue entrenar el modelo a diferentes tamaños de imagen. Los tamaños seleccionados para experimentar en esta fase fueron 800, 960 y 1024. Estos valores se eligieron después de comprobar si es divisible para 32.

La importancia de realizar esta fase fue para conocer el impacto de las resoluciones de las imágenes para el tiempo de entrenamiento, tiempo de inferencia y el rendimiento del modelo.

### 2.3.9. Entrenamiento con el modelo extra large

Todos los experimentos realizados previamente se realizaron con la versión large del modelo RT-DETR que cuenta con 32 M parámetros y 673 capas, sin embargo, la librería de ultralytics tiene disponible la versión extra large del modelo RT-DETR. Esta versión tiene 67 M parámetros y 867 capas que a diferencia del modelo large se duplica el número de parámetros empleados. Se entrenó el modelo extra large para analizar las diferencias de rendimiento con el modelo large.

### 2.3.10. Resumen de entrenamientos

Todo el proceso de entrenamiento se resume en la Tabla 17.

**Tabla 17**

*Resumen de entrenamientos realizados por cada versión del dataset*

| Fase de entrenamiento | Dataset v1 | Dataset v2 | Dataset v3 | Dataset v4 |
|---|---|---|---|---|
| Baseline | Aplicado | Aplicado | Aplicado | Aplicado |
| Optimizadores | No aplicado | Aplicado | Aplicado | Aplicado |
| Modo tuneo | No aplicado | Aplicado | No aplicado | Aplicado |
| Ajuste de hiperparámetros | No aplicado | Aplicado | No aplicado | Aplicado |
| Escalado de imagen | No aplicado | Aplicado | No aplicado | Aplicado |
| Modelo extra large | No aplicado | Aplicado | No aplicado | Aplicado |

*Fuente:* Autoría propia.

Cada una de las versiones del dataset de malezas influyó directamente el avance del entrenamiento del modelo. Por ejemplo, la versión uno, al ser un dataset sin la clase “otros”, fue descartada directamente para continuar con las experimentaciones. De la misma manera, la versión tres fue un dataset con las clases completamente desbalanceadas y se descartó en su momento. Las únicas versiones que completaron el proceso de experimentaciones fueron la versión dos y tres, debido a su nivel aceptable de calidad datos.

## 2.4. Desarrollo de la aplicación web

El desarrollo de la aplicación web es una parte importante para facilitar el uso del modelo entrenado por cualquier tipo de usuario. Se eligió desarrollar una aplicación web debido a su gran compatibilidad con la mayoría de los sistemas operativos de la actualidad, dado que solo se requiere un navegador con conexión a internet para su acceso.

El aplicativo se desarrolló bajo la clásica arquitectura cliente–servidor donde, el modelo se alojó en el lado del servidor y este expone una API REST para facilitar su consumo. Por otro lado, el cliente se encarga de comunicarse con el servidor para realizar las predicciones y visualizar los resultados al usuario. Todo este sistema se empaquetó en contenedores de docker para facilitar el transporte y difusión del software.

### 2.4.1. Servidor

Se desarrolló el servidor con el framework minimalista Flask escrito en python, se optó por esta solución dado que se puede integrar fácilmente el modelo de inteligencia artificial al estar escrito en los mismos lenguajes de programación.

Como se muestra en la Tabla 18, simplemente se emplearon dos rutas para que el usuario pueda acceder a la aplicación web y predecir las malezas haciendo una petición a la ruta "/predict".

**Tabla 18**

*Rutas implementadas en el lado del servidor*

| Ruta | Método | Propósito |
|---|---|---|
| "/" | GET | Mostrar la página principal al usuario. |
| "/predict" | POST | Leer la imagen enviada por el usuario, predecir las malezas y enviar los resultados de inferencia y la imagen predicha en un fichero comprimido (.zip). |

*Fuente:* Autoría propia.

Con la finalidad de ofrecer mayor flexibilidad al usuario a la hora de predecir las malezas, la ruta "/predict" acepta los parámetros descritos en la Tabla 19.

**Tabla 19**

*Parámetros para ajustar los resultados de inferencia*

| Parámetro | Valor por defecto | Descripción |
|---|---|---|
| conf | 0.25 | Establece el umbral mínimo de confianza para las detecciones. |
| imgsz | 640 | Define el tamaño de imagen para la inferencia. |

| | | |
|---|---|---|
| max_det | 300 | Define el número máximo de detecciones por imagen. |
| augment | False | Habilita el aumento de datos en tiempo de prueba para (TTA) para las predicciones. |
| classes | None | Define las clases permitidas para la visualización. |
| save_crop | False | Guarda los recortes de las predicciones basado en las cajas delimitadoras predichas. |
| save_txt | False | Guarda las detecciones en formato [clase] [x_centro] [y_centro] [ancho] [alto]. |
| save_conf | False | Guarda la confianza de cada predicción en formato [clase] [x_centro] [y_centro] [ancho] [alto] [confianza]. |
| show_labels | True | Muestra la etiqueta de cada detección en la imagen predicha. |
| show_conf | True | Muestra la puntuación de confianza de cada detección en la imagen predicha. |
| line_width | None | Especifica el ancho de las líneas de la caja delimitadora. |

*Fuente:* Adaptado de *(Jocher & Rizwan, 2024)*.

El fichero comprimido enviado por el servidor tiene la estructura de directorios que se muestra en la Tabla 20. El fichero "predict/labels/imagen_predicha.txt" es disponible solo si activa "Guardar resultados en texto" o "Guardar puntuaciones de confianza". También, los recortes de predicción ubicados en "predict/crops/" es disponible solo si activa "Guardar recortes de detección". Estos parámetros son ajustados cuando el usuario realiza la predicción desde la aplicación web.

**Tabla 20**

*Contenidos del fichero comprimido enviado por el servidor*

| **Fichero comprimido** | **Descripción** |
|---|---|
| "inference_results.json" | Contiene todos los resultados de inferencia en formato JSON. |
| "predict/imagen_predicha.jpg" | Imagen predicha por el modelo. |

| | |
|---|---|
| “predict/crops/” | Directorio que contiene los recortes de detecciones de cada objeto separadas por clases. |
| “predict/labels/imagen_predicha.txt” | Fichero en texto plano que solo contiene las detecciones en formato [clase] [x_centro] [y_centro] [ancho] [alto] [confianza]. |

*Fuente:* Autoría propia.

### 2.4.2. Cliente

Se utilizó la biblioteca React para el desarrollo de la aplicación web en el lado del cliente, esta librería permite crear interfaces de usuario dinámicas a partir de componentes. El rol principal que el cliente realiza es facilitar al usuario en el proceso de predicción de las malezas. La interfaz de la aplicación web se compone de tres secciones que son: analizar imagen, resultados e historial.

En la Figura 22 se presenta el apartado de analizar imagen, donde el usuario puede subir una imagen, ajustar los parámetros descritos en la Tabla 19 y realizar la predicción. Adicionalmente se agregó tres imágenes de prueba y testear el funcionamiento de la aplicación.

**Figura 22**

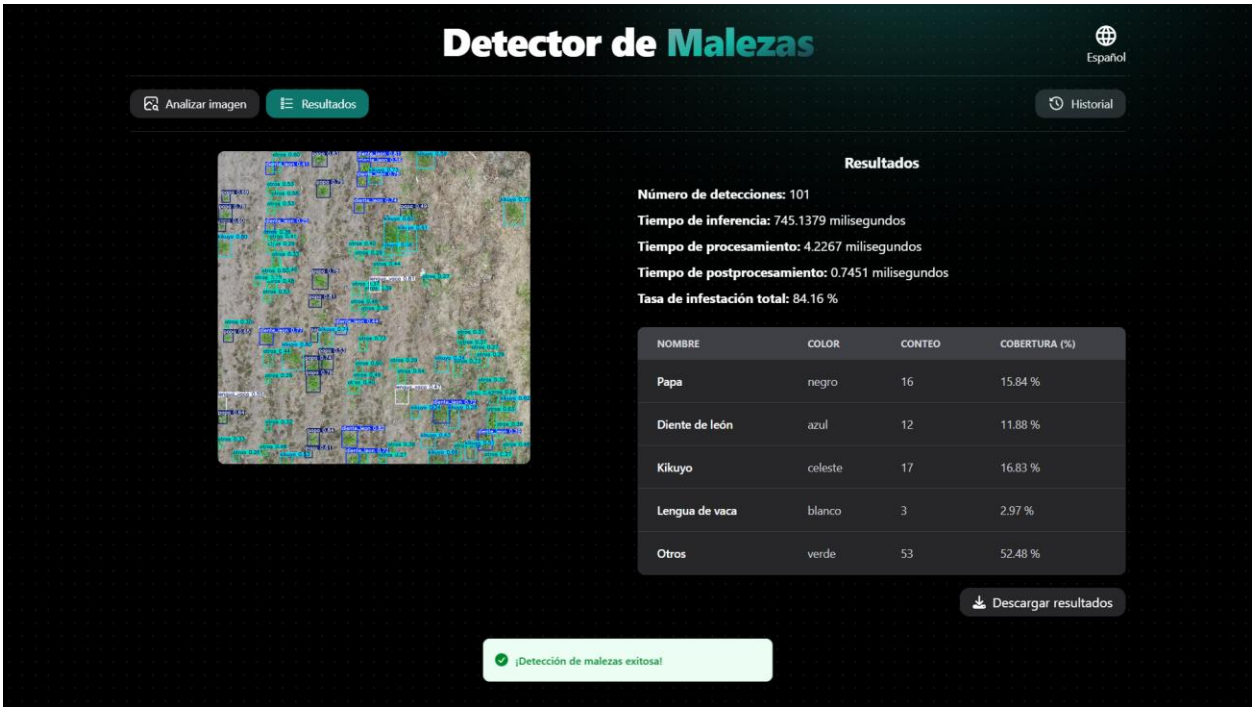
*Sección analizar imagen de la aplicación web*

*Fuente:* Autoría propia.

La sección de resultados simplemente se encarga de mostrar las predicciones al usuario como se observa en la Figura 23. Los resultados son presentados en la parte derecha donde se informa el número total de detecciones, tiempos de inferencia y porcentaje de infestación. Luego, se expone mediante una tabla las estadísticas de conteo y porcentaje de cobertura por cada tipo de planta predicha. Finalmente, se incluyó un botón para que el usuario pueda descargar los resultados obtenidos.

**Figura 23**

*Sección resultados de la aplicación web*




*Fuente:* Autoría propia.

La última sección de la aplicación web es el historial, este apartado tiene la finalidad listar todas las predicciones hechas por el usuario. Como se visualiza en la Figura 24, esta sección únicamente se compone de una tabla paginada que presenta el id, nombre de la imagen y fecha de predicción por cada uno de los registros. En la columna acciones se implementó tres botones para ver el registro, descargar los resultados del registro y eliminar el registro. Finalmente, se agregó un buscador de registros y un botón para limpiar todos los registros.

**Figura 24**

*Sección historial de la aplicación web*


*Fuente:* Autoría propia.

### 2.4.3. Contenedores de desarrollo

La tecnología de docker se utilizó tanto para el desarrollo como para la producción de la aplicación web. Se implementaron los famosos contenedores de desarrollo (dev containers) de Visual Studio Code, que hacen uso de la tecnología docker compose para montar un proyecto dentro de un contenedor. Simplemente se emplea un archivo llamado “devcontainer.json” y la configuración de servicios de docker compose en la raíz del proyecto. Este contenedor es un espacio aislado del sistema anfitrión que permite ejecutar aplicaciones, depuradores o separar herramientas, bibliotecas, extensiones en tiempo real (Microsoft, 2024).

## 2.5. Integración del modelo Transformer en la aplicación web

La integración del modelo Transformer en la aplicación web consiste en utilizar el conocimiento final que fue adquirido después de haber entrenado el modelo de inteligencia artificial de acuerdo con la metodología KDD.

### 2.5.1. Empaquetamiento de la aplicación web

En el empaquetamiento de la aplicación web, nuevamente se utilizó la tecnología de Docker para definir el lenguaje de programación y todas las dependencias necesarias para arrancar la aplicación web.

La imagen de Docker primero describe la versión de Python base del proyecto; luego, se compiló el código del cliente en modo producción; después, se instaló las dependencias del servidor; también, se copió el código compilado del cliente como recurso estático en el servidor para, finalmente, poner en marcha la aplicación web con Gunicorn en el puerto 5000.

Para concretar el proceso de empaquetamiento, se eligió el mejor modelo obtenido para realizar predicciones en el servidor y se construyó la imagen del proyecto con la etiqueta "aluissp/weed-detector:0.1.0".

### 2.5.2. Despliegue

Para poner en línea la aplicación web se eligió el servicio "App Services" de la plataforma Azure debido a su compatibilidad con Docker. Otra plataforma importante que se usó fue Docker Hub para alojar el software empaquetado en internet. Previo al despliegue de la aplicación web en Azure, se subió la imagen del proyecto a docker hub.

Se siguió el siguiente proceso para desplegar la aplicación web:

a) Elegir el servicio de "App Services" en Azure.
b) Definir el nombre de la aplicación web.
c) Seleccionar contenedor como recurso a publicar en el sistema operativo Linux.
d) Elegir un plan de precios de acuerdo con los recursos de hardware a utilizar.
e) Ingresar las credenciales de Docker Hub.
f) Iniciar el proceso de despliegue.

El proceso de despliegue tomó algunos minutos en concretarse. Una vez finalizado, Azure expone un dominio público para acceder a la aplicación web.

# CAPÍTULO III

# RESULTADOS

## 3.1. Métricas de rendimiento

El proceso de evaluación/validación de un modelo de inteligencia artificial es una parte fundamental para garantizar la calidad del modelo y verificar si cumple con el propósito establecido. En este proceso, se utiliza las métricas de rendimiento obtenidas con el conjunto de datos de validación para evaluar su desempeño.

### 3.1.1. Métricas para detección de objetos

En el contexto de la detección de objetos, existen varias métricas para evaluar el rendimiento de un modelo. Las principales métricas usadas en el presente trabajo son: mAP, mAP@50:95, precision y recall para comparar el rendimiento entre los modelos entrenados. También, se utilizó las gráficas como: precision-recall curve, precision-confidence curve, recall-confidence curve y la matriz de confusión para interpretar el comportamiento del modelo.

Para tener una mejor comprensión de las fórmulas aplicadas en el cálculo de las diferentes métricas, es importante definir como se interpreta cuando una detección es válida o no.

- **True Positive (TP):** Cuando se realiza una detección correcta de un objeto anotado.
- **False Positive (FP):** Cuando se realiza una detección incorrecta de un objeto inexistente o una detección mal ubicada.
- **False Negative (FN):** Cuando un objeto anotado no es detectado.

Es importante mencionar que para las tareas de detección de objetos no se considera las detecciones True Negative (TN), esto es debido a que se puede detectar un número infinito de cajas delimitadores que no deberían ser detectados en una imagen (Padilla et al., 2020).

Una detección es considerada cuando el IoU (Intersection Over Union) es mayor o igual que el umbral fijado, caso contrario, se descarta la detección debido a que no cumple con el nivel de solapamiento requerido sobre el objeto detectado.

La precisión (P) se puede definir como la capacidad del modelo para detectar correctamente los objetos encontrados. Como se observa en la ecuación 3, la precisión es el porcentaje de predicciones positivas correctas (Padilla et al., 2020).

$$P = \frac{TP}{TP + FP} = \frac{TP}{all\ detections} \tag{3}$$

La capacidad del modelo para detectar todos los objetos etiquetados se puede definir como recall (R). La ecuación 4 define el recall como el porcentaje de predicciones positivas correctas entre las etiquetas anotadas (Padilla et al., 2020).

$$R = \frac{TP}{TP + FN} = \frac{TP}{all\ ground\ truths} \tag{4}$$

El comportamiento del modelo se puede generalizar con las métricas mAP y mAP@50:95. La diferencia entre estas métricas es el umbral de confianza establecido para el solapamiento de la detección sobre el objeto anotado (IoU). La métrica mAP establece el promedio de la precisión (AP) para todas las clases bajo el umbral de IoU de 0.5. Sin embargo, mAP@50:95 calcula el promedio de la precisión media en un rango de umbrales de IoU que va desde 0.5 hasta 0.95, con incrementos de 0.05 (Padilla et al., 2020).

En la ecuación 5, se observa la definición de la métrica mean average precision (mAP) que calcula el promedio de la precisión media para cada clase, considerando un umbral de intersección sobre unión de 0.5. Cabe destacar que la AP se calcula con la interpolación continua para todos los puntos.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{5}$$

### 3.1.2. Fase 1 – Baseline

El entrenamiento de modelo baseline se aplicó para todas las versiones de los dataset, como se observa previamente en la Tabla 17. En la Tabla 21, se observan las métricas alcanzadas.

**Tabla 21**

*Métricas del modelo baseline*

| Versión del dataset | Precision | Recall | mAP | mAP@50:95 |
|---|---|---|---|---|
| Dataset v1 | **0.828** | **0.799** | **0.842** | **0.511** |
| Dataset v2 | 0.732 | 0.725 | 0.740 | 0.412 |
| Dataset v3 | 0.665 | 0.661 | 0.661 | 0.376 |
| Dataset v4 | 0.763 | 0.777 | 0.796 | 0.439 |

*Nota:* Las métricas obtenidas corresponden a cada una de las versiones del dataset que fueron utilizados. Autoría propia.

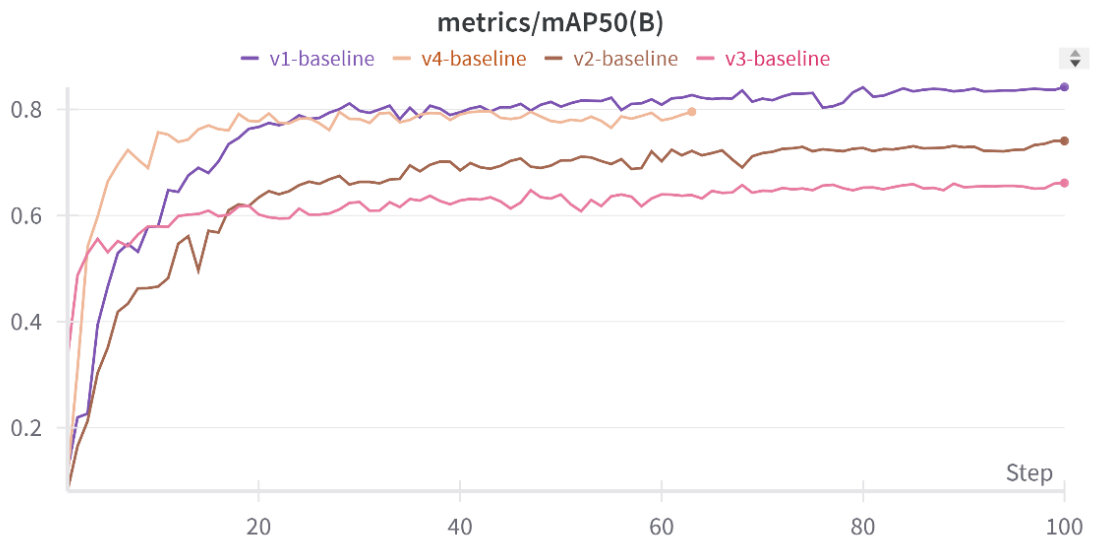
Según la Tabla 21, las mejores métricas alcanzadas fue con la primera versión del dataset con 0.842 mAP y 0.511 mAP@50:95. Este relativo éxito se puede atribuir a que esta versión del dataset no considera la clase “otros”, dado que, detectar esta clase supone un desafío para los modelos. Las plantas anotadas en esta clase representan a especies que no se pudo identificar visualmente como una categoría específica o son especies diferentes que no se tuvo en cuenta como una clase individual.

En la Figura 25, se puede observar el entrenamiento del modelo baseline, donde destaca el rendimiento del modelo para las versiones 1 y 4 del dataset. La versión 4 del dataset consigue sobresalir sobre las versiones 2 y 3 gracias a su nivel balanceado de clases y número de imágenes. Esto quiere decir que el número de muestras por clases, sin sobrerrepresentación, influye bastante en el rendimiento del modelo.

Como en las versiones 2 y 3 no se tiene un ajuste correcto de clases, se obtuvo métricas significativamente inferiores respecto a las demás. La versión 3 del dataset obtuvo la peor métrica, con 0.661 mAP y 0.376 mAP@50:95. A pesar de ser el dataset con más ejemplares anotados, se observa un desempeño pobre debido a la sobrerrepresentación de la clase “papa”, que está casi cinco veces más representada que las otras clases.

**Figura 25**

*Métrica mAP del modelo baseline*




*Nota:* La figura ilustra la métrica mean average precision del modelo baseline para cada una de las versiones del dataset utilizados. Autoría propia.

### 3.1.3. Fase 2 – Optimizadores

Una vez fijado las métricas de rendimiento del modelo baseline, se experimentó el modelo con los optimizadores Adam, RAdam, NAdam y SGD. Se descartó la primera versión del dataset por la falta de la clase "otros". A continuación, se analizaron los resultados obtenidos por cada una de las versiones del dataset.

**Resultados del dataset versión 2**

Este dataset consta de cinco clases: papa, diente de león, kikuyo, lengua de vaca y otros. En la Tabla 22, se observa las métricas alcanzadas por cada optimizador mencionado previamente.

**Tabla 22**

*Métricas de rendimiento bajo distintos optimizadores para la segunda versión del dataset*

| Optimizador | Precision | Recall | mAP | mAP@50:95 |
|---|---|---|---|---|
| Adam | 0.748 | **0.723** | **0.743** | **0.414** |
| NAdam | **0.751** | 0.697 | 0.727 | 0.407 |

| | | | | |
|---|---|---|---|---|
| RAdam | 0.704 | 0.697 | 0.707 | 0.393 |
| SGD | 0.552 | 0.404 | 0.353 | 0.194 |

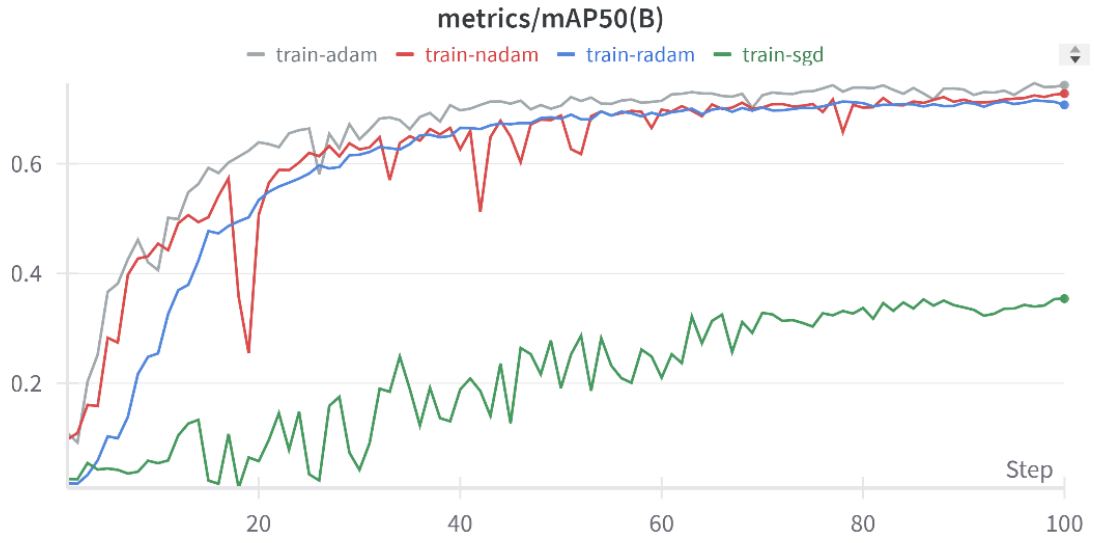
*Nota:* Los resultados se obtuvieron simplemente cambiando el optimizador y se mantuvieron las métricas del modelo baseline. Autoría propia.

De acuerdo a los resultados obtenidos, el mejor modelo se obtuvo con el optimizador Adam, con 0.743 mAP y 0.414 mAP@50:95. Esto representa una mejora con respecto al modelo baseline de la versión 2 del dataset, lo que sugiere que Adam es el optimizador más adecuado para esta versión. Cabe destacar que el optimizador NAdam alcanzó una precisión de 0.751, lo que indica un buen rendimiento en términos de predicciones correctas. Sin embargo, no logró detectar todos los casos, generando un mayor número de falsos negativos (FN).

En la Figura 26 se visualizan los resultados de la métrica mAP para cada optimizador. La familia de optimizadores Adam presenta un rendimiento relativamente similar. Sin embargo, el optimizador SGD tiene un desempeño pobre debido a que requiere más tiempo de entrenamiento (número de épocas) y un dataset relativamente grande para mostrar buenos resultados.

**Figura 26**

*Métrica mAP bajo distintos optimizadores de la versión 2 del dataset*




*Nota:* Autoría propia.

**Resultados del dataset versión 3**

La tercera versión del dataset también fue sometida en la experimentación con los distintos optimizadores. Los resultados obtenidos se muestran en la Tabla 23.

**Tabla 23**

*Métricas de rendimiento bajo distintos optimizadores para la tercera versión del dataset*

| Optimizador | Precision | Recall | mAP | mAP@50:95 |
|---|---|---|---|---|
| Adam | 0.646 | **0.652** | 0.639 | **0.363** |
| NAdam | 0.648 | 0.634 | 0.631 | 0.359 |
| RAdam | **0.653** | 0.646 | **0.644** | 0.361 |
| SGD | 0.518 | 0.546 | 0.511 | 0.273 |

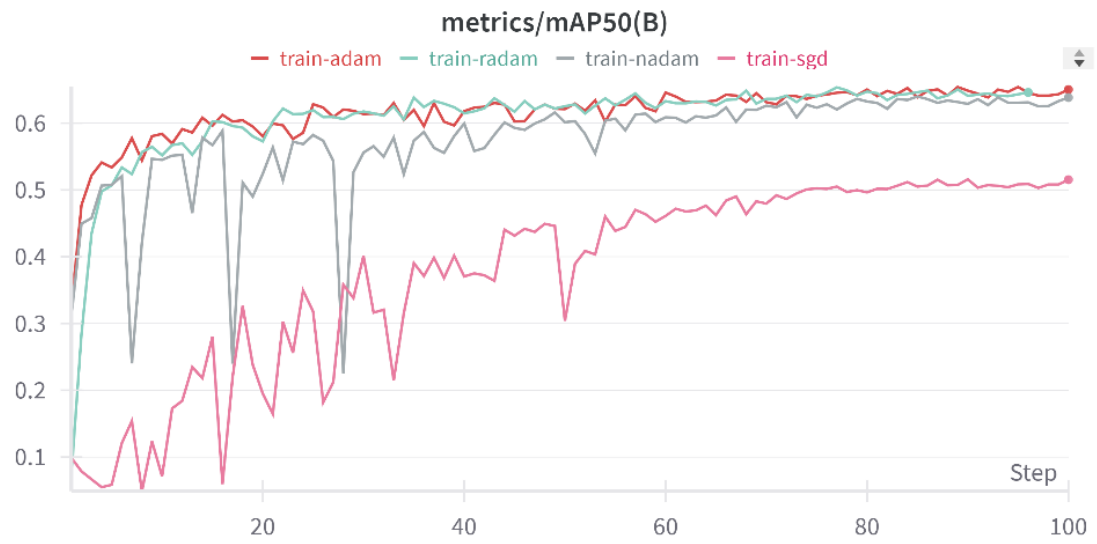
*Nota:* Los resultados se obtuvieron simplemente cambiando el optimizador y se mantuvieron las métricas del modelo baseline. Autoría propia.

Los resultados obtenidos en esta versión no son muy favorables. Los mejores resultados se obtuvieron con el optimizador RAdam con 0,653 de precisión y 0,644 de mAP, mientras que con el optimizador Adam se obtuvieron 0,652 de recall y 0,363 de mAP@50:95.

En comparación con las métricas del modelo baseline y el resto de los optimizadores, no se lograron mejoras significativas, como se observa en la Figura 27. Además, se detectaron dificultades de aprendizaje del modelo con los optimizadores SGD y NAdam, puesto que el aprendizaje comenzó a estabilizarse entre las épocas 50 y 60. Esto sugiere que la versión 3 del dataset definidamente debe ser descartada para futuras pruebas debido a su gran desbalance de clases (papa y el diente de león).

**Figura 27**

*Métrica mAP bajo distintos optimizadores de la versión 3 del dataset*




*Fuente:* Autoría propia.

**Resultados del dataset versión 4**

En la Tabla 24, se visualiza los resultados obtenidos de la cuarta versión del dataset.

**Tabla 24**

*Métricas de rendimiento bajo distintos optimizadores para la cuarta versión del dataset*

| Optimizador | Precision | Recall | mAP | mAP@50:95 |
|---|---|---|---|---|
| Adam | 0.772 | **0.748** | 0.788 | **0.440** |
| NAdam | 0.776 | 0.739 | 0.787 | 0.433 |
| RAdam | **0.795** | 0.723 | **0.793** | 0.430 |
| SGD | 0.724 | 0.715 | 0.744 | 0.398 |

*Nota:* Los resultados se obtuvieron simplemente cambiando el optimizador y se mantuvieron las métricas del modelo baseline. Autoría propia.

En esta fase de experimentación no se obtuvo mejoras considerables. Sin embargo, de acuerdo con los resultados obtenidos con el optimizador RAdam, este modelo mejoró ligeramente la precisión con una puntuación de 0.795, lo que representa un incremento de 0.032 con respecto

al modelo baseline. Esto indica que el modelo evita realizar falsos positivos (FP), es decir, evita predecir objetos inexistentes, pero sacrifica la capacidad de recordar todos los objetos etiquetados.

La Figura 28 ilustra la métrica mAP durante el entrenamiento del modelo. Se destaca es el alto nivel de convergencia de los optimizadores RAdam y Adam, cuyos resultados son casi similares, en ambas curvas alcanzan un buen rendimiento al inicio y finalizan su entrenamiento alrededor de la época 50 gracias a la parada temprana (Early Stopping). No obstante, el optimizador NAdam presenta cierta variabilidad en progreso del entrenamiento, con picos y caídas que demuestran una inestabilidad del modelo. Finalmente, el optimizador SGD tiene un progreso lento y a partir de la época 60 la curva se suaviza, pero con un desempeño muy inferior en comparación con los otros optimizadores.

**Figura 28**

*Métrica mAP bajo distintos optimizadores de la versión 4 del dataset*


*Fuente:* Autoría propia.

### 3.1.4. Fase 3 – Modo tuneo

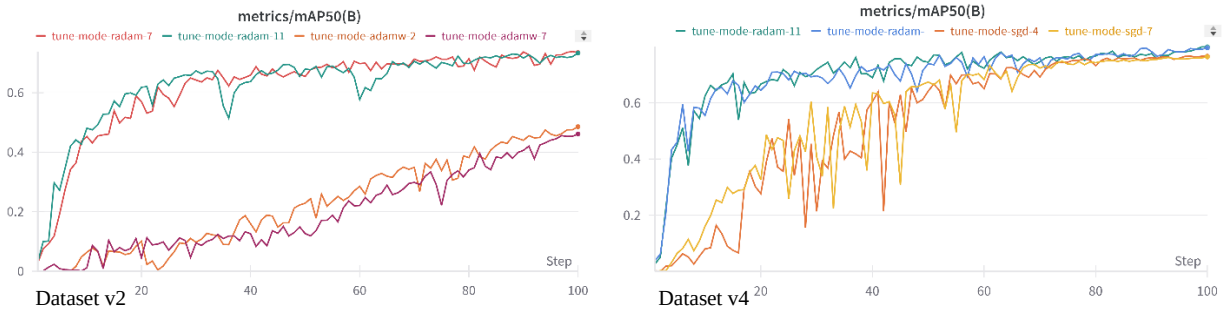
A partir de esta fase, únicamente se consideraron las versiones 2 y 4 del dataset de malezas. Esto se debe a que con la segunda versión se realizó todas las fases de entrenamiento y la cuarta versión fue la que se consiguió balancear las clases al final.

La mejor métrica obtenida fue de 0.734 mAP para el dataset 2 y de 0.799 mAP para el dataset 4, ambas con el optimizador RAdam. Los resultados fueron relativamente similares a los del modelo baseline. Sin embargo, es importante destacar que el modo automatizado de optimización de hiperparámetros resultó ser muy costoso en términos de tiempo y recursos de computación. Para obtener los resultados mencionados, fue necesario ajustar el parámetro de iteraciones a 13 por entrenamientos de 100 épocas, con una parada temprana de 20. Dado que se trata de optimizaciones automáticas, no todos los experimentos realizados por iteración tienen un buen rendimiento, lo que implica un costo de cómputo innecesario por excesivas horas.

Si se busca obtener buenos resultados utilizando este modo, se deben destinar enormes cantidades de tiempo de cómputo. Por ejemplo, establecer el número de iteraciones en 500 o 1000 implica que estos algoritmos necesiten varios días de entrenamiento continuo para evolucionar adecuadamente. A continuación, se analizará la línea de entrenamiento de algunos de estos modelos en ambos datasets.

**Figura 29**

*Métrica mAP del modo de optimización automático de hiperparámetros*




*Nota:* A la izquierda de visualiza algunas de las iteraciones realizadas para el optimizador RAdam y AdamW en el dataset v2 y a la derecha se observa las iteraciones realizadas para el optimizador RAdam y SGD en el dataset v4. Autoría propia.

En la Figura 29 se puede observar el progreso de algunas de las iteraciones realizadas para ajustar los hiperparámetros del dataset v2 y v4. A simple vista, se aprecia una buena convergencia en ciertos casos, pero también se presentan muchos picos y caídas irregulares que provocan inestabilidad en el modelo. Se puede deducir que el optimizador RAdam se adapta moderadamente bien a los hiperparámetros generados por el algoritmo de tuneo, sin embargo, como se observa en el dataset v2, los hiperparámetros ofrecidos al optimizador AdamW pueden resultar muy caóticos

y generar un rendimiento por debajo de 0.5 mAP. En el caso del dataset v4, también existe dificultad para que el optimizador SGD converja, y apenas a partir de la época 70 se puede observar una curva más suavizada.

### 3.1.5. Fase 4 – Ajuste de hiperparámetros

El ajuste de los hiperparámetros fue una fase crucial en el entrenamiento del modelo de inteligencia artificial. Se experimentó con los hiperparámetros que se muestran en la Tabla 16, aunque para cada entrenamiento se alteraron tanto el optimizador como la tasa de aprendizaje. En este caso, se utilizaron los optimizadores Adam, AdamW y RAdam debido a su buen desempeño en las fases previas. Por otro lado, se experimentó con seis diferentes variaciones de la tasa de aprendizaje: 0.005, 0.001, 0.0001 y tres adicionales calculadas a partir de la (2).

**Resultados del dataset versión 2**

El ajuste de los hiperparámetros se realizó 15 veces para la segunda versión del dataset. En la Tabla 25 se observa los resultados alcanzados.

**Tabla 25**

*Métricas de rendimiento del ajuste de hiperparámetros de la segunda versión del dataset*

| Optimizador | Tasa de aprendizaje | Precision | Recall | mAP | mAP@50:95 |
|---|---|---|---|---|---|
| Adam | 0.005 | 0.6625 | 0.6871 | 0.6957 | 0.3904 |
| Adam | 0.001 | 0.7343 | 0.7191 | 0.7357 | 0.4104 |
| Adam | 0.0001 | 0.6707 | 0.6694 | 0.6686 | 0.3640 |
| Adam | 0.0025 | 0.7240 | 0.7160 | 0.7355 | 0.4138 |
| Adam | 0.001111 | **0.7463** | 0.7104 | 0.7359 | **0.4155** |
| Adam | 0.000714 | 0.7342 | 0.7180 | 0.7336 | 0.4080 |
| AdamW | 0.005 | 0.7337 | 0.7236 | 0.7365 | 0.4115 |
| AdamW | 0.001 | 0.7347 | 0.7115 | 0.7329 | 0.4085 |
| AdamW | 0.0001 | 0.6903 | 0.6589 | 0.6673 | 0.3598 |

| | | | | | |
|---|---|---|---|---|---|
| AdamW | 0.0025 | 0.7425 | 0.7092 | **0.7419** | 0.4152 |
| AdamW | 0.001111 | 0.7156 | 0.7269 | 0.7345 | 0.4091 |
| AdamW | 0.000714 | 0.7087 | **0.7377** | 0.7300 | 0.4052 |
| RAdam | 0.005 | 0.7431 | 0.7255 | 0.7418 | 0.4119 |
| RAdam | 0.001 | 0.7071 | 0.7026 | 0.7100 | 0.3912 |
| RAdam | 0.0001 | 0.5347 | 0.4966 | 0.4878 | 0.2673 |

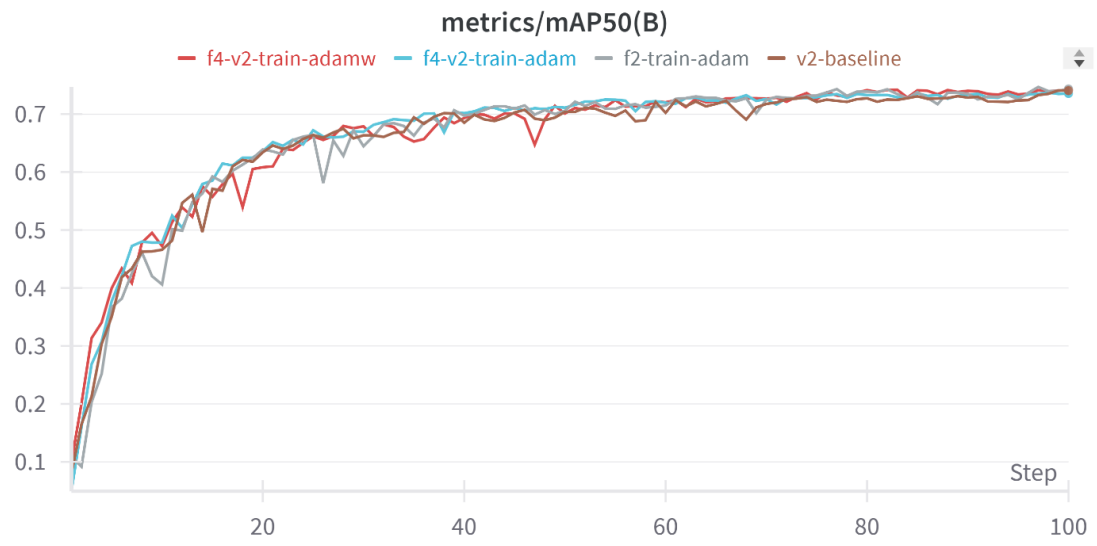
*Nota:* Se experimentó solo con los optimizadores Adam y AdamW para el ajuste de la tasa de aprendizaje descrita en la (2). Autoría propia.

De acuerdo con los resultados obtenidos, la mejor métrica fue de 0.7419 mAP, alcanzada con el optimizador AdamW y una tasa de aprendizaje de 0.0025. Sin embargo, al compararla con los 0.743 mAP obtenidos en la fase 2, no se observaron mejoras significativas; la diferencia fue de solo 0.0011. No obstante, al considerar la métrica mAP@50:95, se obtuvo un puntaje de 0.4155 con el optimizador Adam y una tasa de aprendizaje de 0.001111, superando ligeramente al modelo baseline y al modelo de la fase 2 en 0.0015 y 0.0035 puntos, respectivamente.

Gracias a la Figura 30, se puede realizar una comparación general del rendimiento de los modelos baseline, fase 2 y fase 4. La gráfica muestra una curva de aprendizaje similar para cada modelo, y a partir de la época 50, todos superan los 0,7 de mAP, convergiendo lentamente hasta las 100 épocas. Cabe destacar que, a pesar de tener rendimientos similares, los modelos entrenados con los optimizadores Adam (fase 2) y AdamW (fase 4) presentan picos y caídas al inicio del entrenamiento hasta alcanzar la época 50. En contraste, el modelo baseline y el modelo de la fase 4 entrenado con Adam muestran curvas más suaves desde el inicio hasta el final del aprendizaje.

**Figura 30**

*Métrica mAP de los modelos baseline, fase 2 y 4 de la versión 2 del dataset*




*Nota:* El prefijo "fn" hace referencia a la fase n del modelo. Autoría propia.

**Resultados del dataset versión 4**

En la Tabla 26, se observa los resultados de los 15 experimentos realizados en el dataset v4.

**Tabla 26**

*Métricas de rendimiento del ajuste de hiperparámetros de la cuarta versión del dataset*

| Optimizador | Tasa de aprendizaje | Precision | Recall | mAP | mAP@50:95 |
|---|---|---|---|---|---|
| Adam | 0.005 | 0.7776 | 0.7433 | 0.7832 | 0.4325 |
| Adam | 0.001 | 0.7483 | 0.7590 | 0.7899 | 0.4352 |
| Adam | 0.0001 | 0.7747 | 0.7508 | 0.7910 | 0.4278 |
| Adam | 0.0025 | 0.7655 | 0.7682 | 0.7933 | 0.4352 |
| Adam | 0.001111 | 0.7733 | 0.7610 | 0.7926 | **0.4383** |
| Adam | 0.000714 | 0.7676 | 0.7449 | 0.7920 | 0.4343 |

| | | | | | |
|---|---|---|---|---|---|
| AdamW | 0.005 | 0.7908 | 0.7627 | 0.7926 | 0.4377 |
| AdamW | 0.001 | **0.8085** | 0.7280 | 0.7912 | 0.4377 |
| AdamW | 0.0001 | 0.7715 | 0.7624 | 0.7978 | 0.4299 |
| AdamW | 0.0025 | 0.7767 | 0.7681 | 0.7970 | 0.4354 |
| AdamW | 0.001111 | 0.7459 | **0.7713** | 0.7869 | 0.4360 |
| AdamW | 0.000714 | 0.7884 | 0.7574 | **0.7990** | 0.4353 |
| RAdam | 0.005 | 0.7290 | 0.7541 | 0.7758 | 0.4195 |
| RAdam | 0.001 | 0.7963 | 0.7324 | 0.7853 | 0.4330 |
| RAdam | 0.0001 | 0.7436 | 0.7241 | 0.7578 | 0.4080 |

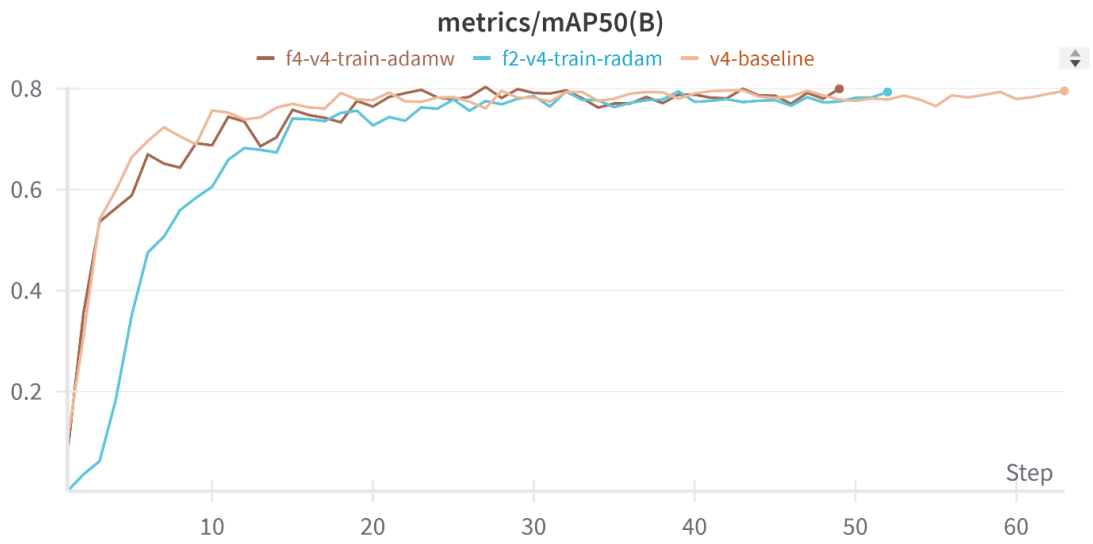
*Nota:* Se experimentó solo con los optimizadores Adam y AdamW para el ajuste de la tasa de aprendizaje con la ecuación 2. Autoría propia.

En la versión 4 del dataset, se logró mejorar el rendimiento del modelo, alcanzando un mAP de 0.799, en comparación con los 0.793 mAP de la fase 2 y los 0.796 mAP del modelo baseline. Esto representa una diferencia de 0.006 y 0.003 puntos, respectivamente. El optimizador AdamW, con una tasa de aprendizaje entre 0.001 y 0.0001 (de 0.000714), se ajusta razonablemente para la predicción y el recuerdo correcto de objetos anotados, obteniendo una precisión de 0.7884 y un recall de 0.7574.

No obstante, si se considera la métrica mAP@50:95, se obtuvo un valor de 0.4383, inferior a los 0.439 del modelo baseline y a los 0.440 de la fase 2. Esto indica que, al considerar los distintos niveles de solapamiento (IoU), no se obtuvieron mejoras en esta fase. Otra métrica interesante es la precisión, que alcanzó un valor de 0.8085 con el optimizador AdamW y una tasa de aprendizaje de 0.001. Este es el mejor valor de precisión obtenido hasta el momento, pero a costa de un sacrificio notable en el recall, que disminuyó hasta 0.7280, lo cual indica un buen desempeño en la predicción de objetos, pero a expensas de omitir varios objetos anotados.

**Figura 31**

*Métrica mAP de los modelos baseline, fase 2 y 4 de la versión 4 del dataset*




*Nota:* El prefijo "fn" hace referencia a la fase n del modelo. Autoría propia.

La Figura 31 presenta una comparación entre el modelo baseline y los modelos de las fases 2 y 4 para la cuarta versión del dataset. En general, se observa que los modelos presentan una convergencia similar, aunque con algunas oscilaciones, y finalizan el entrenamiento antes de las 100 épocas. El modelo baseline muestra un comportamiento estable desde el inicio, con una curva suave, pero finaliza su entrenamiento después de 60 épocas. El modelo de la fase 2 presenta una curva suave, pero su aprendizaje inicial es lento. El modelo de la fase actual exhibe un comportamiento similar al baseline, con ligeras caídas, pero finaliza el entrenamiento antes que los demás.

### 3.1.6. Fase 5 – Escalado de imagen

Todos los entrenamientos realizados previamente fueron a 640 píxeles de resolución de imagen. En esta fase se experimentó aumentando el tamaño a 800px, 960px y 1120/1024px con los mejores hiperparámetros obtenidos en las fases anteriores.

**Resultados del dataset versión 2**

El experimento que se realizó para esta versión del dataset fue aplicar el escalado de imagen para los optimizadores Adam y AdamW con los mejores hiperparámetros obtenidos en la fase 4 como se muestra en la Tabla 27.

**Tabla 27**

*Métricas de rendimiento del escalado de imagen de la segunda versión del dataset*

| Optimizador | Tamaño de imagen | Tasa de aprendizaje | Precision | Recall | mAP | mAP@50:95 |
|---|---|---|---|---|---|---|
| Adam | 800 | 0.001111 | 0.751 | 0.731 | 0.758 | 0.432 |
| Adam | 960 | 0.001111 | 0.749 | 0.757 | 0.764 | 0.444 |
| Adam | 1120 | 0.001111 | 0.755 | **0.763** | **0.768** | **0.447** |
| AdamW | 800 | 0.0025 | 0.751 | 0.737 | 0.755 | 0.430 |
| AdamW | 960 | 0.0025 | **0.766** | 0.743 | 0.766 | 0.443 |
| AdamW | 1120 | 0.0025 | 0.731 | 0.758 | 0.763 | 0.445 |

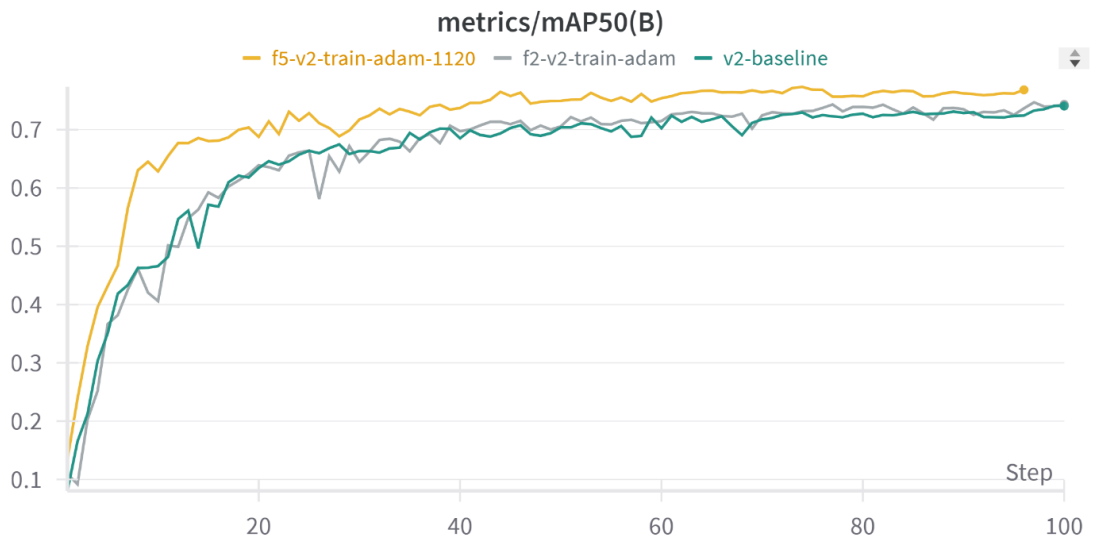
*Fuente:* Autoría propia.

El escalado de imagen mejoró considerablemente las métricas obtenidas en el modelo baseline y de la fase 2. En este caso, el optimizador Adam con una resolución de 1120px y 0.001111 como tasa de aprendizaje consigue obtener 0.763 de recall, 0.768 de mAP y 0.447 de mAP@50:95. Si se compara con el puntaje de 0.740 y 0.743 de mAP para el modelo baseline y el modelo de la fase 2, representa unos 0.028 y 0.025 puntos de diferencia, respectivamente.

Cabe destacar que este modelo también alcanza un buen puntaje en las métricas de recall y mAP@50:95, donde se puede entender que el modelo es suficientemente capaz de detectar todas las plantas anotadas y también es tolerante a los diferentes niveles IoU. Sin embargo, es importante mencionar que las diferencias no son abismales si lo comparamos con las otras resoluciones, y dependiendo del caso de uso se podría apostar por un modelo entrenado con menos píxeles.

**Figura 32**

*Métrica mAP de los modelos baseline, fase 2 y 5 de la versión 2 del dataset*




*Nota:* El prefijo "fn" hace referencia a la fase n del modelo. Autoría propia.

Según la Figura 32, se puede destacar la rápida convergencia del modelo de color amarillo con algunas leves caídas. En general, a partir de la época 40 no se observa mejoras de aprendizaje, que incluso en algunas épocas los modelos olvidaban el conocimiento aprendido. A partir de la gráfica también se puede interpretar que el aumento de la escala de la imagen ayuda a obtener mejores métricas y a una rápida convergencia al inicio del entrenamiento, gracias a que el modelo puede aprender las características representativas de cada clase.

**Resultados del dataset versión 4**

A diferencia de los experimentos realizados para el dataset versión 2, en este dataset se realizó más pruebas con el escalado de imagen y los hiperparámetros utilizados, como se visualiza en la Tabla 28.

**Tabla 28**

*Métricas de rendimiento del escalado de imagen de la cuarta versión del dataset*

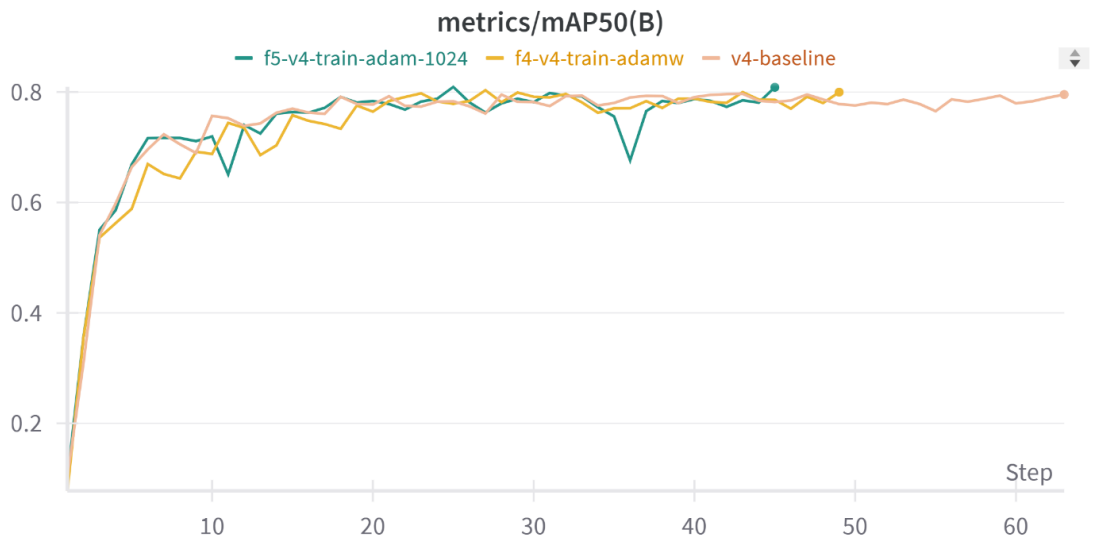
| Opt. | Hiperparámetros aplicados | Tasa de aprendizaje | Tamaño de imagen | Precision | Recall | mAP | mAP@50:95 |
|---|---|---|---|---|---|---|---|
| Adam | Baseline | 0.001111 | 800 | 0.7929 | 0.7602 | 0.8003 | 0.4487 |
| Adam | Baseline | 0.001111 | 960 | 0.7643 | 0.7560 | 0.7868 | 0.4429 |
| Adam | Baseline | 0.001111 | 1024 | 0.7762 | 0.7701 | 0.7953 | 0.4423 |
| Adam | Fase 5 | 0.0025 | 800 | 0.7838 | 0.7677 | 0.7999 | 0.4378 |
| Adam | Fase 5 | 0.0025 | 960 | 0.7535 | 0.7706 | 0.7823 | 0.4313 |
| Adam | Fase 5 | 0.0025 | 1024 | 0.7628 | **0.7867** | 0.7998 | 0.4400 |
| Adam | Fase 5 | 0.001111 | 800 | 0.7595 | 0.7495 | 0.7872 | 0.4385 |
| Adam | Fase 5 | 0.001111 | 960 | 0.7628 | 0.7798 | 0.8083 | **0.4489** |
| Adam | Fase 5 | 0.001111 | 1024 | 0.7860 | 0.7734 | **0.8092** | 0.4482 |
| Adam | Fase 5 | 0.000714 | 800 | 0.7894 | 0.7518 | 0.7940 | 0.4418 |
| Adam | Fase 5 | 0.000714 | 960 | **0.8032** | 0.7427 | 0.7931 | 0.4394 |
| Adam | Fase 5 | 0.000714 | 1024 | 0.7579 | 0.7710 | 0.7976 | 0.4461 |
| AdamW | Fase 5 | 0.000714 | 800 | 0.7748 | 0.7604 | 0.7929 | 0.4415 |
| AdamW | Fase 5 | 0.000714 | 960 | 0.7689 | 0.7733 | 0.7934 | 0.4417 |
| AdamW | Fase 5 | 0.000714 | 1024 | 0.7682 | 0.7625 | 0.7978 | 0.4468 |

*Nota:* La columna de hiperparámetros aplicados hacer referencia al resto de hiperparámetros no mencionadas en las columnas y que fueron extraídas del modelo baseline o de la fase 5. Autoría propia.

Conforme a las métricas obtenidas en la Tabla 28, se obtuvo 0.8092 de mAP como la mejor métrica, utilizando el optimizador Adam con una tasa de aprendizaje de 0.001111 y una resolución de 1024 píxeles. En contraste con la métrica del modelo baseline 0.796 mAP, supone una mejora del 0.0132 puntos; y en comparación con la mejor métrica de la fase 5 de 0.7990 mAP, se incrementó en 0.0102 puntos de rendimiento. A pesar de que los resultados obtenidos no sean enormes, si se ajustan correctamente los hiperparámetros y el tamaño de la imagen, se puede mejorar el rendimiento del modelo.

**Figura 33**

*Métrica mAP de los modelos baseline, fase 4 y 5 de la versión 4 del dataset*




*Nota:* El prefijo "fn" hace referencia a la fase n del modelo. Autoría propia.

Conforme a la Figura 33, los tres mejores modelos convergen igual y rápidamente al inicio; luego, la curva se afina a partir de la época 30. El modelo de la fase actual consigue llegar sobre los 0.8 mAP alrededor de la época 25. Sin embargo, este modelo tiene una caída notable después del pico más alto alcanzado; luego, se recupera y termina primero su entrenamiento. Este comportamiento se puede interpretar como que el modelo pierde el conocimiento alcanzado después de la época 25 y termina su entrenamiento después de 20 épocas, gracias a la parada temprana para evitar el desajuste del modelo.

### 3.1.7. Fase 6 – Modelo extra-large

El entrenamiento del modelo extra large es una fase adicional que se realizó para observar si se puede mejorar el rendimiento si se entrena un modelo con muchos más parámetros.

**Resultados del dataset versión 2**

La segunda versión del dataset fue sometido al entrenamiento con el modelo extra-large, y los resultados se visualizan en la siguiente tabla.

**Tabla 29**

*Métricas de rendimiento del modelo extra large de la segunda versión del dataset*

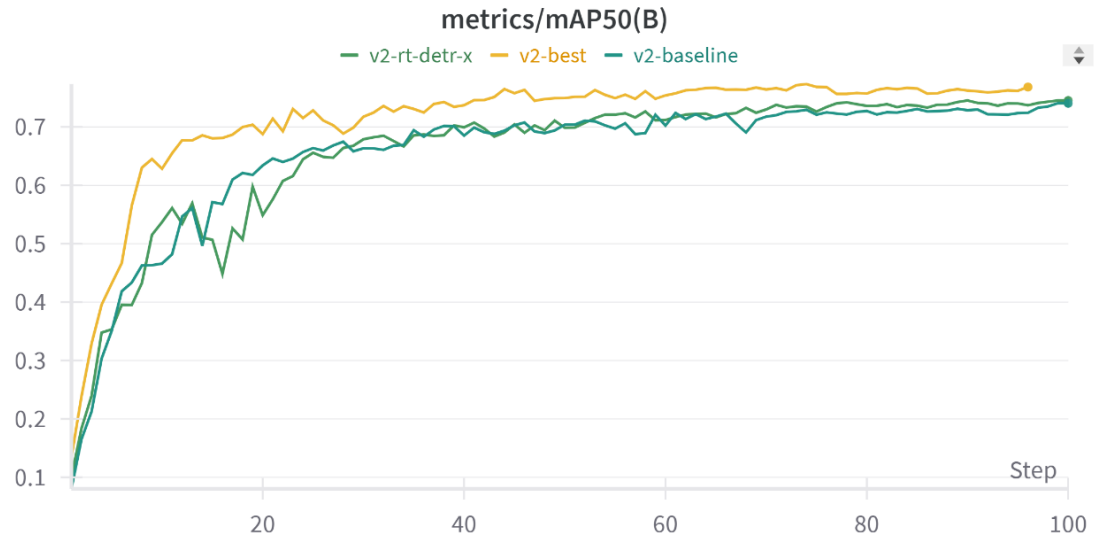
| Versión del modelo | Optimizador | Tamaño de imagen | Precision | Recall | mAP | mAP@50:95 |
|---|---|---|---|---|---|---|
| baseline | AdamW | 640 | 0.732 | 0.725 | 0.740 | 0.412 |
| best | Adam | 1120 | **0.755** | **0.763** | **0.768** | **0.447** |
| extra-large | Adam | 640 | 0.748 | 0.721 | 0.745 | 0.417 |

*Fuente:* Autoría propia.

En la Tabla 29 se comparan las métricas de los modelos baseline, extra-large y el mejor modelo, con las siguientes métricas 0.740, 0.745 y 0.768 de mAP, respectivamente. A breves rasgos, se consiguió un rendimiento muy similar al modelo baseline con la versión extra-large, y por lo tanto no supone grandes diferencias de rendimiento cuando se aumenta el número de parámetros. Además, si el modelo es ajustado correctamente, se pueden simular las métricas alcanzadas del mejor modelo.

**Figura 34**

*Métrica mAP de los modelos baseline, extra-large y el mejor modelo de la versión 2 del dataset*




*Fuente:* Autoría propia.

Estas afirmaciones también se puede corroborar con la Figura 34, donde se observa un aprendizaje similar al modelo baseline, solo con la diferencia de que se tiene de una caída moderada al inicio del entrenamiento, pero no está ni cerca de alcanzar al mejor modelo alcanzado con la versión 2 del dataset.

**Resultados del dataset versión 4**

La cuarta versión del dataset también fue sometido a esta fase de experimentación. En la Tabla 30, se observan los resultados alcanzados.

**Tabla 30**

*Métricas de rendimiento del modelo extra large de la cuarta versión del dataset*

| Versión del modelo | Optimizador | Tamaño de imagen | Precision | Recall | mAP | mAP@50:95 |
|---|---|---|---|---|---|---|
| baseline | AdamW | 640 | 0.763 | **0.777** | 0.796 | 0.439 |
| best | Adam | 1024 | **0.786** | 0.773 | **0.8092** | **0.4482** |
| extra-large | Adam | 640 | 0.762 | 0.774 | 0.792 | 0.436 |

*Fuente:* Autoría propia.

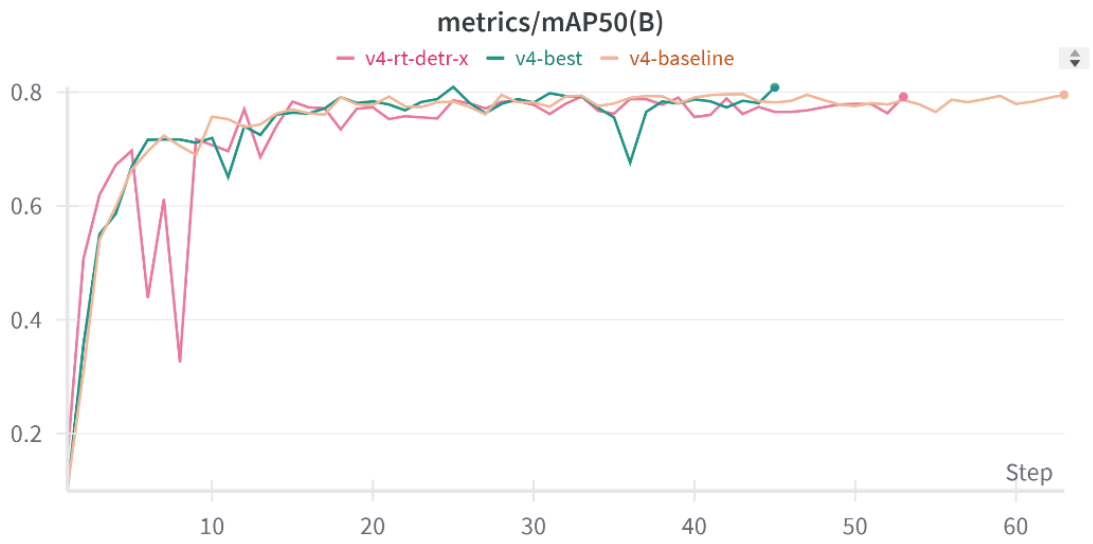
En cuanto a la métrica alcanzada por el modelo extra-large de 0.792 mAP, es muy cercano obtenido por el modelo baseline de 0.796 mAP. Si se compara con el mejor modelo, la diferencia es solo de 0.017 puntos.

Según la Figura 35, a pesar de que el modelo extra-large tenga puntajes similares de mAP, al inicio del entrenamiento tiene una caída bastante pronunciada hasta la época 10, lo que puede deberse a la fase de calentamiento aplicado inicio. Sin embargo, poco a poco se va suavizando la curva y termina el entrenamiento alrededor de la época 50, al no detectar mejoras significativas hace 20 épocas previas.

En general, el entrenamiento con el modelo extra-large demostró tener resultados muy similares al modelo baseline en los dataset 2 y 4, lo que significa que a pesar de que el modelo RT-DETR aumente su número de parámetros, esto no garantizará resultados superiores frente al modelo baseline.

**Figura 35**

*Métrica mAP de los modelos baseline, extra-large y el mejor modelo de la versión 4 del dataset*




*Fuente:* Autoría propia.

### 3.1.8. Evaluación del mejor modelo alcanzado

Antes de definir el mejor modelo obtenido, se realizó un breve análisis de las mejores métricas obtenidas en cada fase (exceptuando la fase 3 y 6) para los datasets de la versión 2 y 4, en los términos de la métrica mAP y mAP@50:95. Luego, se realizó la interpretación de comportamiento del mejor modelo.

**Tabla 31**

*Mejores métricas de los dataset versión 2 y 4*

| Nombre de la fase | Dataset v2 | | Dataset v4 | |
|---|---|---|---|---|
| | mAP | mAP@50:95 | mAP | mAP@50:95 |
| Baseline | 0.740 | 0.412 | 0.796 | 0.439 |
| Optimizadores | 0.743 | 0.414 | 0.793 | 0.440 |
| Ajuste de hiperparámetros | 0.7419 | 0.4155 | 0.7990 | 0.4383 |

| | | | | |
|---|---|---|---|---|
| Escalado de imagen | **0.768** | **0.447** | **0.8092** | **0.4482** |

*Nota:* La tabla refleja las mejores métricas alcanzadas en negrita. Autoría propia.

La diferencia de rendimiento que se consiguió con ambos dataset es notable, según la Tabla 31. La versión 4 del dataset logró rebasar en 0.0412 puntos en términos de la métrica mAP a la segunda versión; probablemente este comportamiento se deba al buen balanceo de clases y número de imágenes de la versión 4. Además, tomando de referencia a las métricas del modelo baseline, cuando se avanza el entrenamiento en las fases de optimizadores y el ajuste de los hiperparámetros, las métricas mejoran ligeramente en ambos datasets.

Los resultados de la fase de escalado de imagen superan notablemente a las métricas del modelo baseline. Por ejemplo, si se observan las métricas mAP del dataset versión 2, se denota un estancamiento alrededor de los 0.74 mAP; pero cuando se somete a la fase de escalado de imagen, el modelo mejora hasta los 0.76 mAP de rendimiento. Estos resultados prueban que el tamaño de las imágenes tiene un impacto muy importante para obtener un buen desempeño. Sin embargo, si tiene en cuenta a la versión 4 del dataset no se alcanzó el mismo impacto que en la versión 2.

Si se analiza la evolución del rendimiento en la versión 4 del dataset, se entiende que cuando se consigue la métrica del modelo baseline, se disminuye ligeramente el rendimiento en la fase de los optimizadores; luego, se mejora el rendimiento en la fase de ajuste hiperparámetros donde incluso se llega a aproximar a los 0.8 puntos de mAP, pero con el escalado de imagen se logra romper esta barrera, consiguiendo el puntaje de 0.8092 mAP.

Los resultados obtenidos quieren decir que no siempre se podrá mejorar el modelo simplemente cambiando el optimizador para lograr ver mejores resultados. También es importante ajustar los hiperparámetros y, por otra parte, la resolución de las imágenes ayuda notablemente al rendimiento del modelo

#### 3.1.8.1. Análisis de los mejores resultados

Discutir sobre cuál es el mejor modelo alcanzado puede resultar ser un poco difícil, ya que se deben considerar diferentes aristas para definir cuál es el modelo que mejor se desempeña en resolver el problema planteado. Una vía que se puede seguir es utilizar las mejores métricas alcanzadas para determinar el mejor modelo.

De acuerdo a la Tabla 31, las mejores métricas se alcanzaron con los modelos obtenidos del dataset versión 4. Para esta evaluación, se seleccionó el mejor modelo en cuanto a la métrica mAP y otro basado en la métrica mAP@50:95.

A continuación, se presentan las métricas de rendimiento, las épocas y el tiempo de entrenamiento obtenidos de los dos mejores modelos para realizar esta evaluación, en la Tabla 32.

**Tabla 32**

*Métricas de rendimiento de los mejores modelos basados en mAP y mAP@50:95*

| Versión del modelo | Precision | Recall | mAP | mAP@50:95 | Épocas | Tiempo |
|---|---|---|---|---|---|---|
| best-1 | **0.7860** | 0.7734 | **0.8092** | 0.4482 | 45 | 39’49” |
| best-2 | 0.7628 | **0.7798** | 0.8083 | **0.4489** | 73 | 60’32” |

*Fuente:* Autoría propia.

Los modelos presentados en la tabla anterior tienen demasiada similitud en cuanto al rendimiento general. Sin embargo, existe una notable diferencia en el tiempo de entrenamiento empleado. El modelo best-1 demuestra una mejor convergencia, puesto que solo emplea 39 min 49 s de entrenamiento en 45 épocas, es decir, 20 min 43 s menos que el modelo best-2 donde se empleó 60 min con 32 s en 73 épocas.

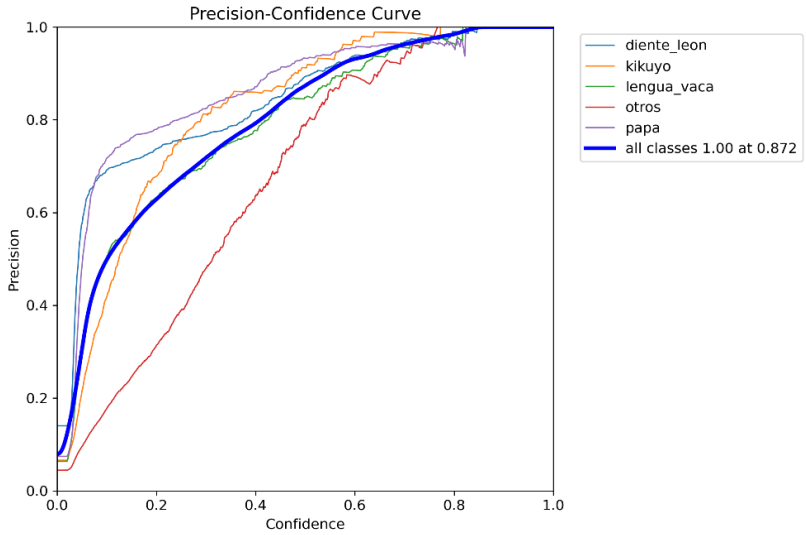
### 3.1.8.2. Curvas de rendimiento

A continuación, se emplearon las distintas curvas de rendimiento para denotar mejor las diferencias de rendimiento entre ambos modelos.

De acuerdo a la Figura 36 y Figura 37, ambos modelos tienen un buen desempeño para detectar las clases de papa, diente de león y kikuyo, es decir, que a confianzas bajas el modelo es capaz de predecir correctamente estas clases. Por el contrario, si se tiene en cuenta la clase de lengua de vaca, el modelo best-1 muestra un rendimiento intermedio, pero en el caso del modelo best-2, tiene una precisión por debajo del desempeño global. En ambos casos, se muestra una baja capacidad para detectar la clase otros, es decir, los modelos realizan más predicciones incorrectas a medida que la confianza disminuye.

**Figura 36**

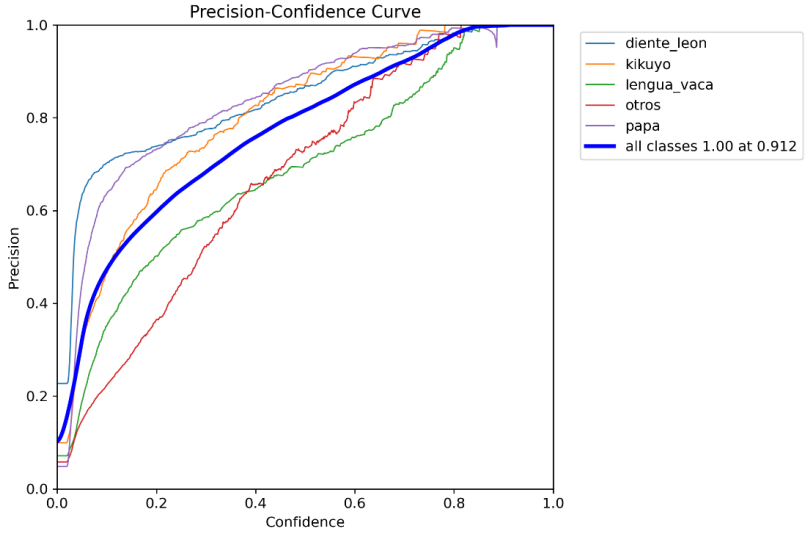
*Curva de precision-confidence de modelo best-1*




*Fuente:* Autoría propia.

**Figura 37**

*Curva de precision-confidence de modelo best-2*



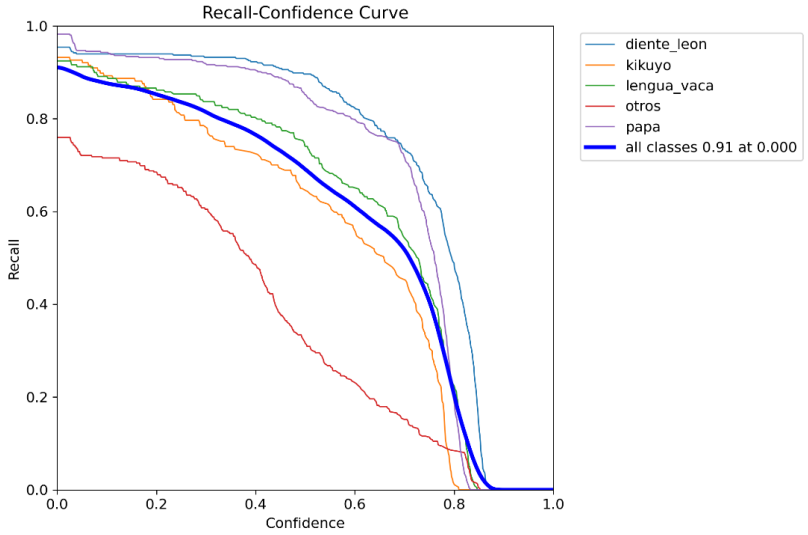

*Fuente:* Autoría propia.

Para entender la Figura 38 y Figura 39, se debe leerlas desde derecha a izquierda, puesto que estas métricas son ordenadas por el valor de confianza de forma descendente. El modelo best-2 muestra un recall (recuerdo) rápido para las clases papa, lengua de vaca y diente de león. En

ambos casos, la clase kikuyo es ligeramente inferior al rendimiento global. La detección de la clase otros es muy similar, pero se observa que el modelo best-1 supera levemente al modelo best-2. Estos comportamientos evidencian perfectamente puntaje obtenido en la Tabla 32.

**Figura 38**

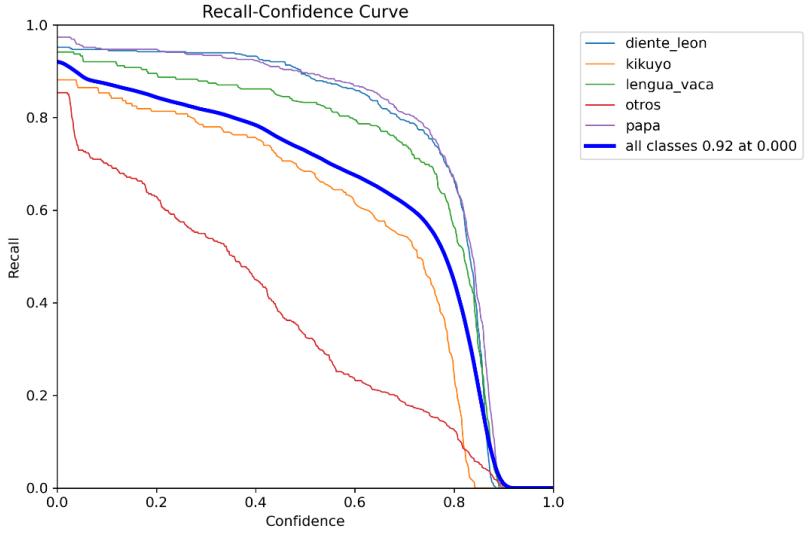
*Curva de recall-confidence de modelo best-1*




*Fuente:* Autoría propia.

**Figura 39**

*Curva de recall-confidence de modelo best-2*



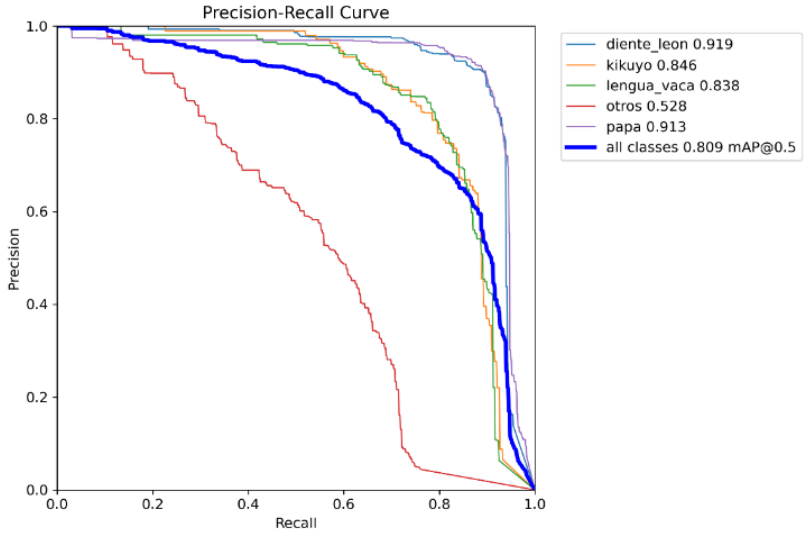
*Fuente:* Autoría propia.

La gráfica de compensación entre las métricas precision y recall se muestran en la Figura 40 y Figura 41 para los modelos best-1 y best-2, respectivamente. A nivel general, ambos modelos tiene un buen equilibrio rendimiento para las clases: papa, diente de león, kikuyo y lengua de vaca con puntuaciones superiores de 0.8 mAP.

No obstante, la clase otros es la más problemática de todas debido a su caótico conjunto de datos con diferentes especies de plantas que esta abarca. Si se observa las gráficas, esta clase penaliza demasiado el rendimiento modelo, con puntuaciones cercanas a 0.8 mAP en ambos modelos.

**Figura 40**

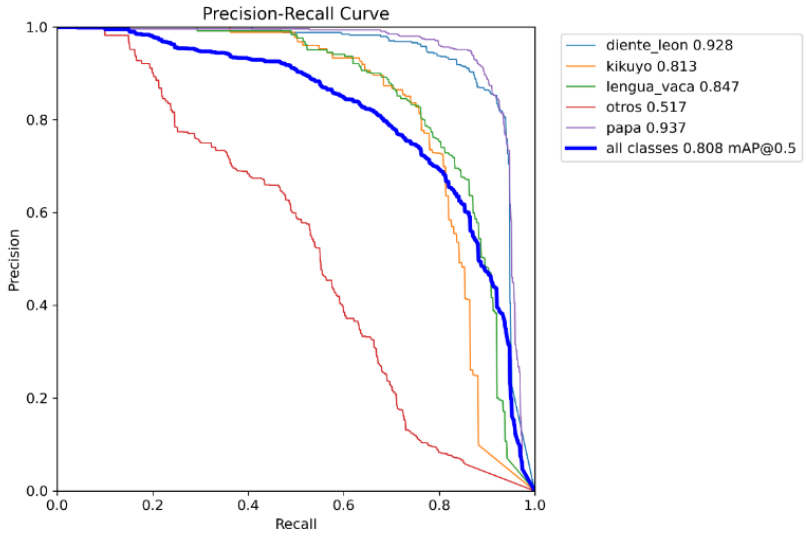
*Curva de precision-recall de modelo best-1*




*Fuente:* Autoría propia.

**Figura 41**

*Curva de precision-recall de modelo best-2*




*Fuente:* Autoría propia.

### 3.1.8.3. Matriz de confusión

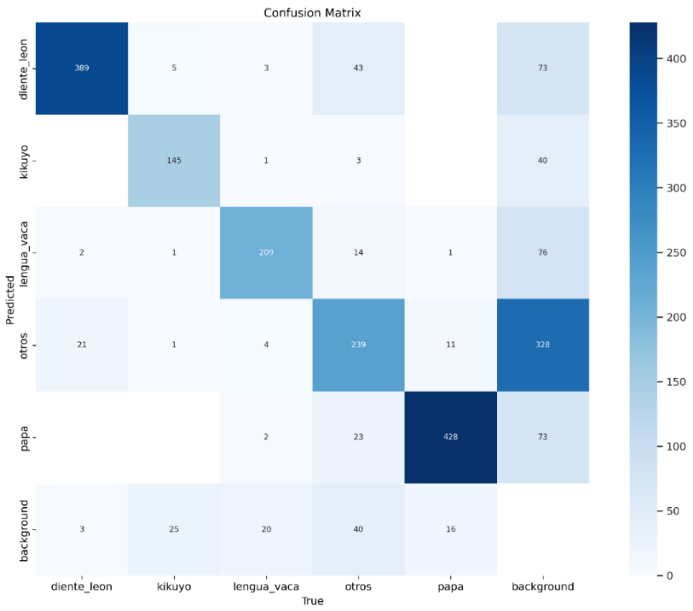
Y por último, se examinó el comportamiento de los modelos a nivel individual empleando la matriz de confusión. En la Figura 42 y Figura 43, se observa las predicciones realizadas sobre el conjunto de datos de validación para el modelo best-1. Como se evidenció previamente, se obtiene buen un rendimiento a nivel general. No obstante, se detecta que el modelo tiende a realizar más predicciones falsas positivas (FP) que falsas negativas (FN), en especial, con la clase otros.

Por ejemplo, el modelo predice 328 instancias incorrectamente que son menos de las 239 instancias predichas correctamente. Esto indica que el modelo es capaz de detectar plantas muy pequeñas que no fueron anotadas, o también puede ser que el modelo detecte plantas donde en realidad no existe nada.

Tomando en cuenta la matriz de confusión normalizada, se observa que las instancias predichas correctamente para la clase otros corresponde solo al 66% del total de instancias etiquetadas, lo que representa una brecha muy alta con las otras clases que superan el 80% de predicciones fácilmente.

**Figura 42**

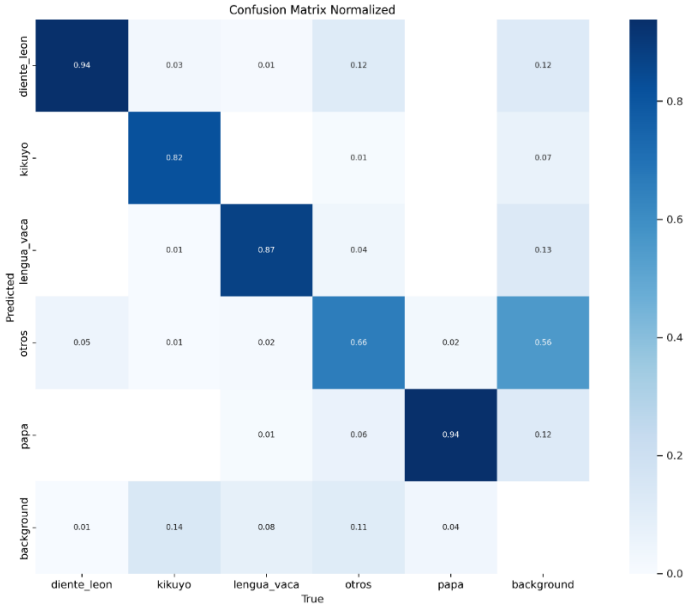
*Matriz de confusión del modelo best-1*




*Fuente:* Autoría propia.

**Figura 43**

*Matriz de confusión normalizada del modelo best-1*



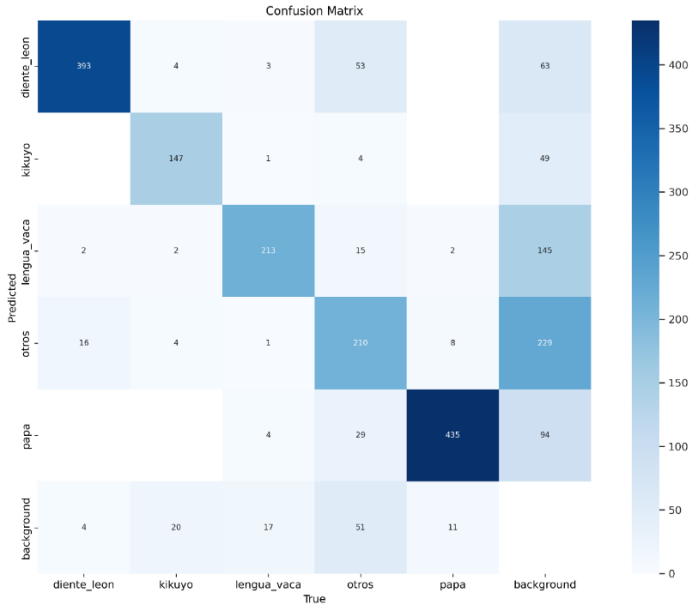

*Fuente:* Autoría propia.

Finalmente, se muestran las predicciones realizadas por el modelo best-2 sobre el conjunto de datos de validación en la Figura 44 y Figura 45. En general, las clases tienen más del 80% de instancias predichas a excepción de la clase otros. De acuerdo a las métricas expuestas en la Tabla 32, este modelo tiene mejor recall que el modelo best-1. Esto posiblemente se puede explicar con el bajo número de predicciones falsas positivas (FP) realizadas para cada clase, puesto que, si se considera la mayor concentración de predicciones incorrectas, se encuentra sobre la columna background donde se realizó 580 predicciones falsas positivas (FP) de un objeto inexistente, 10 predicciones menos que el modelo best-1.

Sin embargo, estas detecciones están repartidas en mayor proporción para las clases lengua de vaca y otros, con 229 y 145 instancias detectadas respectivamente. Al realizar menos predicciones falsas positivas (FP), el modelo debería tener mejor precisión y bajo recall, sin embargo, es todo lo contrario. Este comportamiento puede deberse a la distribución de los falsos positivos para las clases otros y lengua de vaca que, a la hora de calcular las métricas, pueden estar penalizando la precisión sobre el recall.

**Figura 44**

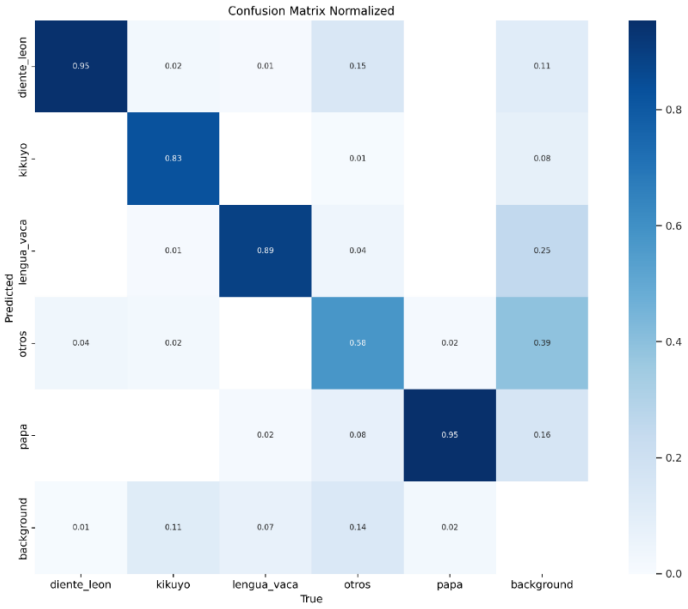
*Matriz de confusión del modelo best-2*




*Fuente:* Autoría propia.

**Figura 45**

*Matriz de confusión normalizada del modelo best-2*




*Fuente:* Autoría propia.

### 3.1.8.4. El mejor modelo alcanzado

Una vez analizado las métricas y curvas de rendimiento, se selecciona al modelo RT-DETR best-1 de la Tabla 32 como el mejor modelo alcanzado en este proyecto. Este modelo no solo se destaca en alcanzar 0.8092 de mAP y 0.4482 de mAP@50:95 a nivel general, si no también, a nivel de las clases individuales alcanza un AP (Average Precisión) superior a los 0.8, con excepción de la clase otros que tiene una puntuación de 0.528 AP en el conjunto de datos de validación.

La Tabla 33 muestra el rendimiento del modelo RT-DETR para los conjuntos de validación y testeo, donde los resultados son casi parejos con una diferencia menor o igual al 6% para la métrica mAP. Esto indica que el modelo logra generalizar bien para localizar plantas nunca antes vistas, esta es una característica muy deseada a la hora de poner el modelo a producción.

**Tabla 33**

*Métricas del mejor modelo RT-DETR alcanzado*

| Versión | Métrica | Conjunto de datos de validación (validation set) | Conjunto de datos de testeo (test set) |
|---|---|---|---|
| RT-DETR best-1 | Papa AP | 0.913 | 0.921 |
| | Kikuyo AP | 0.846 | 0.714 |
| | Diente de león AP | 0.919 | 0.824 |
| | Lengua de vaca AP | 0.838 | 0.856 |
| | Otros AP | 0.528 | 0.493 |
| | Precision | 0.786 | 0.731 |
| | Recall | 0.773 | 0.752 |
| | mAP | 0.8092 | 0.762 |
| | mAP@50:95 | 0.4482 | 0.434 |

*Nota:* Tanto el conjunto de validación como el de testeo solo tienen el 10% del total de datos, es decir, 70 imágenes cada una. Autoría propia.

Sin embargo, es importante mencionar que todo el proceso de selección del mejor modelo fue realizado en base a los resultados obtenidos en el conjunto de datos de validación. Evidentemente, el rendimiento disminuye si se evalúa con el conjunto de datos de testeo, es decir, con datos nunca vistos en el entrenamiento del modelo.

En la Figura 46 se observa una pequeña muestra de imágenes en formato RGB, las etiquetas anotadas por cada clase y las predicciones realizadas por el modelo. Estas imágenes fueron recortadas del tamaño original para visualizar mejor las plantas y etiquetas. Enfocándose en las imágenes anotadas y las detecciones del modelo, se puede observar algunas predicciones adicionales que no fueron etiquetadas por ser plantas muy pequeñas. Este hecho justificaría la alta cantidad de falsos positivos (FP) que se puede observar en la matriz de confusión.

A nivel de producción, se podría valorar mucho más que el modelo detecte precisamente las plantas anotadas individualmente y, con respecto a la clase otros que al ser más genérica abarca

ciertas especies que no se observó en el campo de cultivo al inicio o que no se pudo diferenciar a simple vista, se puede ser más flexivos a la hora de detectarlos.

**Figura 46**

*Muestras de cultivo y malezas reales, anotadas y predichas por el modelo RT-DETR*

*Nota:* La primera fila son imágenes reales de cultivo y malezas, la segunda son imágenes anotadas manualmente y la tercera fila son las predicciones realizadas por el modelo RT-DETR. La papa está representada por el color azul, el diente de león aparece en amarillo, el kikuyo en celeste, la lengua de vaca en rojo y las otras malezas en morado. Autoría propia.

## 3.2. Validación estadística

El problema que se pretende solucionar en el presente trabajo es la dificultad para la detección de malezas en campos de cultivos de papa. Mediante la prueba t-Student, se puede determinar si la cuantificación de las plantas anotadas manualmente discrepan significativamente

de las plantas que predice el modelo. A través de esta evaluación, se determinará si la aplicación web realiza predicciones fiables para resolver la problemática planteada.

Para emplear la prueba t-Student, primero se definió la hipótesis nula ($H_0$) y la hipótesis alternativa ($H_1$) de la siguiente manera:

- $H_0: d = 0$, No hay diferencia significativa entre las medias de la cuantificación de plantas manualmente y las detectadas por la aplicación.
- $H_1: d \neq 0$, Si hay diferencia significativa entre las medias de la cuantificación de plantas manualmente y las detectadas por la aplicación.

Luego, se definió la regla de decisión que establece: si $p - value \leq 0.05$, rechazar la hipótesis nula y aceptar la hipótesis alternativa.

Para realizar esta prueba se empleó el conjunto de datos testing que corresponde al 10% del total de imágenes del dataset y el programa Microsoft Excel. Primero, se realizó un conteo de cada planta por separado de las 70 imágenes del conjunto de testing. Luego, se contó las plantas inferidas por la aplicación web. Una vez recolectados estos datos, se utilizó Excel para realizar la prueba t-Student para dos muestras, asumiendo varianzas iguales. Esta prueba está disponible en el apartado de análisis de datos.

### 3.2.1. Prueba t-Student para la cuantificación de la papa

Los resultados obtenidos tras realizar la prueba t-Student se observa en la Tabla 34. Se obtuvo un valor de $p = 0.688$. Según la regla de decisión, no se rechaza $H_0$, lo que significa que no hay diferencias significativas entre la cuantificación manual y la cuantificación hecha por la inferencia de la aplicación web para la detección de las papas.

**Tabla 34**

*Resultados de la prueba t-Student para la cuantificación de la papa*

| | **Cuantificación de la aplicación web** | **Cuantificación manual** |
|---|---|---|
| Media | 10.414 | 9.585 |
| Varianza | 176.854 | 120.825 |

| | |
|---|---|
| Estadístico t | 0.401 |
| Valor p | **0.688** |

*Fuente:* Autoría propia.

### 3.2.2. Prueba t-Student para la cuantificación de diente de león

Los resultados obtenidos tras realizar la prueba se observa en la Tabla 35. Se obtuvo un valor de $p = 0.922$. De acuerdo a la regla de decisión, no se rechaza $H_0$, esto que indica que no hay diferencias significativas entre la cuantificación manual y la cuantificación por la inferencia de la aplicación web para la detección de la maleza de diente de león.

**Tabla 35**

*Resultados de la prueba t-Student para la cuantificación de diente de león*

| | **Cuantificación de la aplicación web** | **Cuantificación manual** |
|---|---|---|
| Media | 8,3 | 8,2 |
| Varianza | 37.778 | 35.727 |
| Estadístico t | 0.097 | |
| Valor p | **0.922** | |

*Fuente:* Autoría propia.

### 3.2.3. Prueba t-Student para la cuantificación del kikuyo

Los resultados después de aplicar la prueba t-Student se observan en la Tabla 36 y se obtiene un valor de $p = 0.968$. De acuerdo a la regla de decisión, no se rechaza $H_0$, y se determina que no hay diferencias significativas entre ambos métodos de cuantificación para detectar kikuyo.

**Tabla 36**

*Resultados de la prueba t-Student para la cuantificación del kikuyo*

| | **Cuantificación de la aplicación web** | **Cuantificación manual** |
|---|---|---|
| Media | 6.342 | 6.414 |
| Varianza | 119.359 | 108.130 |

| | |
|---|---|
| Estadístico t | -0.039 |
| Valor p | **0.968** |

*Fuente:* Autoría propia.

### 3.2.4. Prueba t-Student para la cuantificación de la lengua de vaca

Después de realizar la prueba t-Student en la cuantificación de la maleza de la lengua de vaca, se obtiene un valor de $p = 0.606$, como se observa en la Tabla 37. Esto indica que no hay diferencia significativa en ambas formas de cuantificación.

**Tabla 37**

*Resultados de la prueba t-Student para la cuantificación de la lengua de vaca*

| | **Cuantificación de la aplicación web** | **Cuantificación manual** |
|---|---|---|
| Media | 4.557 | 3.5 |
| Varianza | 183.177 | 110.427 |
| Estadístico t | 0.516 | |
| Valor p | **0.606** | |

*Fuente:* Autoría propia.

### 3.2.5. Prueba t-Student para la cuantificación de la clase otros

Se realizó la prueba t-Student para comprobar las discrepancias de cuantificación manual y la cuantificación por la inferencia de la aplicación web para la detección de "otros" tipos de malezas. De acuerdo a la Tabla 38, se alcanzó un valor $p = 0.015$, lo que quiere decir que si hay una diferencia significativa entre ambos métodos de cuantificación. En contraste al resto de pruebas, solo la clase otros rechaza la hipótesis nula $(H_0)$, esto puede deberse a la dificultad del modelo para detectar diferentes especies de malezas que no fueron consideradas como una clase individual.

**Tabla 38**

*Resultados de la prueba t-Student para la cuantificación de la clase otros*

| | Cuantificación de la aplicación web | Cuantificación manual |
|---|---|---|
| Media | 10.585 | 14.4 |
| Varianza | 43.115 | 125.634 |
| Estadístico t | -2.456 | |
| Valor p | **0.015** | |

*Fuente:* Autoría propia.

### 3.2.6. Prueba t-Student para la cuantificación de malezas

La cuantificación de malezas que implica el conteo de las clases kikuyo, lengua de vaca, diente de león y otros. Como se observa en la Tabla 39, se obtuvo un valor de $p = 0.330$ una vez aplicado la prueba t-Student. Lo que indica esto es que no hay diferencias significativas entre ambas formas de cuantificación y es fiable las predicciones de la aplicación web.

**Tabla 39**

*Resultados de la prueba t-Student para la cuantificación de malezas*

| | Cuantificación de la aplicación web | Cuantificación manual |
|---|---|---|
| Media | 29.785 | 32.514 |
| Varianza | 231.127 | 315.296 |
| Estadístico t | -0.976 | |
| Valor p | **0.330** | |

*Fuente:* Autoría propia.

### 3.2.7. Prueba t-Student para la cuantificación de las plantas

Finalmente, la cuantificación de las plantas toma en cuenta todas las clases del dataset. Los resultados de la prueba t-Student se muestra en la Tabla 40. Se obtuvo un valor de $p = 0.609$, lo que indica que no hay diferencias significativas entre la cuantificación manual y la cuantificación por la inferencia de la aplicación web.

**Tabla 40**

*Resultados de la prueba t-Student para la cuantificación de las plantas*

| | **Cuantificación de la aplicación web** | **Cuantificación manual** |
|---|---|---|
| Media | 40.2 | 42.1 |
| Varianza | 503.930 | 461.221 |
| Estadístico t | -0.511 | |
| Valor p | **0.609** | |

*Fuente:* Autoría propia.

### 3.2.8. Comparativa con las métricas de rendimiento y pruebas t-Student

Una vez realizado las pruebas t-Student, es importante realizar una comparativa con las métricas de rendimiento para confirmar el desempeño del modelo. En la Tabla 41 se muestran los resultados obtenidos del mejor modelo alcanzado.

**Tabla 41**

*Resultados de las métricas de rendimiento y pruebas t-Student*

| **Clase** | Conjunto de datos de validación | Conjunto de datos de testeo | Prueba t-Student |
|---|---|---|---|
| Papa AP | 0.913 | 0.921 | 0.688 |
| Kikuyo AP | 0.846 | 0.714 | 0.968 |
| Diente de león AP | 0.919 | 0.824 | 0.922 |
| Lengua de vaca AP | 0.838 | 0.856 | 0.606 |
| Otros AP | 0.528 | 0.493 | 0.015 |

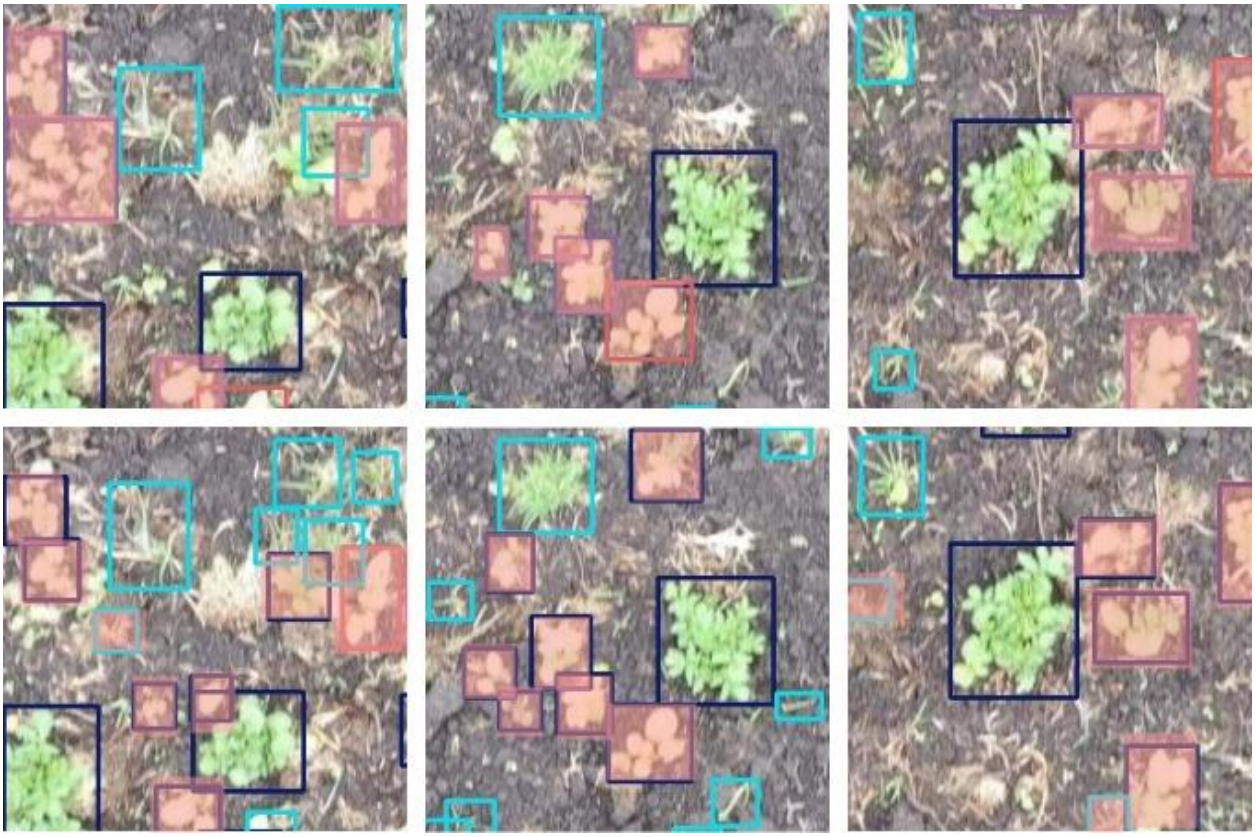
*Fuente:* Autoría propia.

Los resultados obtenidos tanto en las métricas de rendimiento y las pruebas t-Student concuerdan perfectamente. Todas las clases consiguen una precisión media (AP) buena y, consecuentemente, son aceptadas de acuerdo a la prueba t-Student. Sin embargo, la clase “otros” fue la que consiguió el peor rendimiento tanto en los conjuntos de validación como en los testeo,

lo que también se demuestra al realizar la prueba t-Student. Estos resultados pueden deberse a ciertos factores como a la alta presencia de falsos positivos (plantas que no fueron anotadas por no ser identificadas en una clase específica), tamaño, forma de las plantas, presencia de ruido en las imágenes, superposición de plantas y las condiciones de luz desfavorables. En la Figura 47 se ilustran algunas predicciones realizadas por el modelo donde se evidencian estas condiciones inusuales.

**Figura 47**

*Muestra de etiquetas reales y predicciones negativas del conjunto de datos de testeo*



*Nota:* La primera fila corresponden a las etiquetas reales (Ground Truth) y la segunda fila son las predicciones realizadas por el modelo RT-DETR. La papa está representada por el color azul, el diente de león aparece en amarillo, el kikuyo en celeste, la lengua de vaca en rojo y las otras malezas en morado. Todas las predicciones negativas están cubiertas por un cuadro rojo. Autoría propia.

### 3.3. Evaluación con la ISO 25010

En el presente trabajo de titulación se validó la calidad de la aplicación web mediante las características de portabilidad y compatibilidad de la norma ISO 25010.

La evaluación se realizó en los siguientes cuatro sistemas operativos con diferentes configuraciones que se detallan a continuación:

1. Laptop 1 Dell, Windows 11, 32 GB de RAM, 4 GB de GPU y procesador AMD Ryzen 7.
2. Laptop 2 HP, Windows 11, 16 GB de RAM y procesador AMD Ryzen 5.
3. Laptop 3 Dell, Arcolinux, 32 GB de RAM, 4 GB de GPU y un procesador AMD Ryzen 7.
4. Azure App Services, 3.5 GB de RAM, 10 GB de almacenamiento.

### 3.3.1. Medición de portabilidad

En la Tabla 42 se describen las métricas y aplicadas para medir las subcaracterísticas de portabilidad de la aplicación web. Es importante recalcar que se aplicó todos los factores de portabilidad mencionados en dicha norma, con excepción de la capacidad de ser reemplazado. Esta última subcaracterística presupone la existencia de otro software desarrollado previamente con el mismo propósito, lo cual no aplica en este caso. En cuando a la valoración de x, cuanto más se aproxime a 1, mejor será el nivel de portabilidad.

**Tabla 42**

*Métricas y fórmulas aplicadas en la medición de portabilidad*

| Factor de portabilidad | Métrica | Fórmula | Valor de X |
|---|---|---|---|
| Adaptabilidad | Adaptabilidad a distintos dispositivos | $X = 1 - A/B$<br>A= cantidad de dispositivos con problemas de adaptación.<br>B= cantidad de dispositivos en los que el software debe adaptarse. | $0 \leq X \leq 1$ |
| Capacidad para ser instalado | Facilidad de instalación | $X = A/B$<br>A= cantidad de instalaciones satisfactorias por el usuario.<br>B= cantidad de instalaciones totales realizados por el usuario. | $0 \leq X \leq 1$ |

| | | | |
|---|---|---|---|
| | Facilidad de reinstalación | $X = 1 - A/B$<br>A= cantidad de intentos fallidos de reinstalación.<br>B= cantidad total de reinstalaciones. | $0 \leq X \leq 1$ |
| Capacidad para ser reemplazado | Uso continuado de los datos | $X = A/B$<br>A= cantidad de datos reutilizables del software anterior.<br>B= cantidad de datos a reutilizar del software antiguo. | $0 \leq X \leq 1$ |

*Fuente:* Adaptado de (Guamangate et al., 2021).

Para evaluar la adaptabilidad del aplicativo, se ha empleado prueba en varios dispositivos tecnológicos y son valorados como 1 cuando cumple y 0 cuando no aplica, como se ilustra en la Tabla 43.

**Tabla 43**

*Resultados de la evaluación de subcaracterística de adaptabilidad*

| **Criterio** | **Dispositivos** | **Puntaje** |
|---|---|---|
| Responsive | Smart TV | 1 |
| | Ordenador | 1 |
| | Laptop | 1 |
| | Tablet | 1 |
| | Smartphone | 1 |

*Fuente:* Adaptado de (Guamangate et al., 2021).

Una vez obtenido los puntajes de adaptabilidad, se obtiene $B = 5$ y $A = 0$, luego se aplica la fórmula $X = 1 - A/B$ que da como resultado $X = 1 - 0/5 = 1$.

La capacidad para ser instalado se evaluó en dos partes, primero se evaluó la facilidad de instalación por cada uno de los equipos mencionados previamente como se observa en la Tabla 44. Es importante destacar que se usa la tecnología de Docker, lo cual facilita bastante la puesta en marcha del sistema.

**Tabla 44**

*Resultados de la evaluación de subcaracterística de facilidad de instalación*

| Equipo | Intentos fallidos de instalación (A) | Número de intentos (B) |
|---|---|---|
| Laptop 1 | 1 | 1 |
| Laptop 2 | 1 | 2 |
| Laptop 3 | 1 | 1 |
| Azure | 1 | 2 |
| Total | 4 | 6 |

*Fuente:* Autoría propia.

Una vez realizado la evaluación de facilidad de instalación, se aplica la fórmula y se obtiene el siguiente resultado $X = 4/6 = 0.667$.

Finalmente, se evaluó la facilidad de reinstalación, se aplicó esta prueba a los mismos dispositivos como se muestra en la Tabla 45.

**Tabla 45**

*Resultados de la evaluación de subcaracterística de facilidad de reinstalación*

| Equipo | Intentos fallidos de reinstalación (A) | Número de intentos (B) |
|---|---|---|
| Laptop 1 | 0 | 1 |
| Laptop 2 | 0 | 1 |
| Laptop 3 | 0 | 1 |
| Azure | 0 | 1 |
| Total | 0 | 4 |

*Fuente:* Autoría propia.

Cuando se evalúa, se obtiene $A = 0$ y $B = 4$. Luego de aplicar la fórmula, se calcula lo siguiente, $X = 1 - 0/4 = 1$.

Para obtener la portabilidad general de la aplicación web, se debe calcular la media de los puntajes de las subcaracterísticas calculadas como se observa en Tabla 46.

**Tabla 46**

*Cálculo general del nivel de portabilidad de la aplicación web*

| **Subcaracterística de portabilidad** | **Valor (%)** |
|---|---|
| Adaptabilidad de dispositivos | 100 |
| Facilidad de instalación | 66.7 |
| Facilidad de reinstalación | 100 |
| Promedio | 88.9 |

*Fuente:* Autoría propia.

Una vez evaluada las métricas de la característica de portabilidad, se determina una portabilidad del 88.9%, lo que quiere decir es que la aplicación web tiene una alta capacidad de portabilidad. La subcaracterística donde se tiene problemas es en la facilidad de instalación. Esto se debe a que el equipo donde se prueba la instalación es bajo el sistema operativo Windows y, al estar implementando Docker (una tecnología que requiere un subsistema de Linux), no se otorga el 100% de los recursos al contenedor cuando interopera con Windows, lo que puede generar problemas con el subsistema emulado.

### 3.3.2. Medición de compatibilidad

La característica de compatibilidad del sistema fue evaluada de acuerdo a las métricas que se indica en la Tabla 47. En este caso de estudio, se descartó la subcaracterística de interoperabilidad debido a que no se hace uso de sistemas externos, aparte del servidor que está alojado en Azure App Services. En cuando a la valoración de x, cuanto más se aproxime a 1, mejor será el nivel de compatibilidad.

**Tabla 47**

*Métricas y fórmulas aplicadas en la medición de compatibilidad*

| Factor de compatibilidad | Métrica | Fórmula | Valor esperado |
|---|---|---|---|
| Coexistencia | Coexistencia disponible | $X = A/B$<br>A= cantidad de entidades con las que el sistema puede existir.<br>B= cantidad de entidades que deben coexistir en el sistema. | $0 \leq X \leq 1$ |
| Interoperabilidad | Conectividad con sistemas externos | $X = A/B$<br>A= cantidad de interfaces implementadas con sistemas externos.<br>B= total de interfaces con sistemas externas. | $0 \leq X \leq 1$ |
| | Capacidad de intercambio de datos | $X = A/B$<br>A= cantidad de datos transferidos exitosamente a otros sistemas.<br>B= cantidad total de datos que se transfieren. | $0 \leq X \leq 1$ |

*Fuente:* Adaptado de (ISO/IEC 25023, 2016).

Para medir la coexistencia de la aplicación, se puso en marcha la aplicación web junto a 10 aplicaciones de escritorio donde se probó su funcionamiento correcto. Como se observa en la Tabla 48, se calificó como 0 cuando presenta problemas de coexistencia y 1 cuando tiene una coexistencia exitosa.

**Tabla 48**

*Resultados de la evaluación de subcaracterística de coexistencia*

| Tipo de aplicación | Aplicación | Puntuación |
|---|---|---|
| Navegadores | Chrome | 1 |
| | Microsoft Edge | 1 |
| | Brave | 1 |
| | Firefox | 1 |
| Productividad | Microsoft Word | 1 |
| | Microsoft Excel | 1 |
| | Notion | 1 |
| | Microsoft Outlook | 1 |
| Mensajería | WhatsApp versión escritorio | 1 |
| | Telegram versión escritorio | 1 |

*Fuente:* Autoría propia.

Una vez realizado la evaluación, se obtuvo $A = 10$ y $B = 10$, donde se calculó, $X = A/B = 10/10 = 1$. De acuerdo a la evaluación realizada, se determina que la aplicación web coexiste completamente con las aplicaciones cotidianas del día a día. Esto se debe gracias a que no consume muchos recursos para la inferencia de malezas, puesto que la mayoría de las tareas de cómputo se lo realiza en el lado del servidor.

# DISCUSIÓN

En el presente trabajó se consiguió entrenar la arquitectura RT-DETR para la detección automática de malezas en los campos de cultivo de papa. Se eligió este modelo por su rendimiento y destacada capacidad de detección de objetos en tiempo real, que compite directamente con los modelos de vanguardia de la familia YOLO. El modelo fue evaluado mediante las métricas de rendimiento y la validación estadística con pruebas t-Student.

Conforme a la Tabla 33, el mejor modelo alcanzó 0.8092 de mAP y 0.4482 de mAP@50:95 para detectar las clases de: papa, kikuyo, diente de león, lengua de vaca y otros. Estos resultados se obtuvieron gracias al ajuste de los hiperparámetros, el escalado de imagen y, principalmente, a la cuarta versión del dataset. El modelo entrenado con la tercera versión del dataset obtuvo las peores métricas, con 0.661 mAP y 0.376 mAP@50:95, como se observa en la Tabla 21. Este dataset, a pesar de tener más ejemplares anotados, presentó un desempeño pobre debido a la sobrerrepresentación de la clase papa y la anotación de imágenes incorrecta en algunos ejemplares, causada por el solapamiento excesivo del cultivo y las malezas.

La penalización del rendimiento del modelo también se le atribuye a la clase “otros”. Como se observa en las métricas de rendimiento (Tabla 33) y las pruebas t-Student (Tabla 41), se consiguió 0.528 de AP (Average Precision) en la clase “otros”, lo que coincide con el valor $p = 0.015$. Esto evidencia la dificultad del modelo para aprender las diferentes características de las plantas que componen esta clase. Sin embargo, si se excluye la clase “otros” el modelo consigue 0.842 de mAP, como se observa en la Tabla 21. Es decir, el modelo tiende a mostrar mejores resultados cuando solo se enfoca en aprender plantas con clases únicas.

Estos resultados evidencian la importancia de tener una dataset bien ajustado en todas las clases que se desea detectar para conseguir un buen desempeño en el modelo. El de ajuste de hiperparámetros, específicamente el tamaño de las imágenes tiene un gran impacto en el rendimiento del modelo. Como se muestra en la Tabla 31, la fase de escalado de imagen permitió mejorar las métricas de las fases anteriores y conseguir el mejor modelo con 0.8092 mAP de puntuación.

Con respecto al modelo empleado, también se entrenó el modelo RT-DETR-X (67 millones de parámetros). Según la Tabla 30, se consiguió 0.792 mAP, es decir, un rendimiento muy similar

al modelo RT-DET-L (32 millones de parámetros). Esto indica que aumentar el número de parámetros en un modelo Transformers no siempre va a demostrar una mejora significativa de rendimiento, puesto que de por sí ya cuenta con bastantes parámetros.

La comparativa con los trabajos relacionados permite observar el rendimiento del modelo frente a la literatura existente. A continuación, se presentan las comparativas en la Tabla 49.

**Tabla 49**

*Comparativa de modelos Transformer para detección de cultivos y malezas*

| **Autor** | **Modelo** | **Número de clases** | **mAP** |
|---|---|---|---|
| (Abuhani et al., 2023) | DETR | 2 clases: girasoles y malezas. | 0.430 |
| (Deng et al., 2024) | DINO | 8 clases de malezas conocidas. | **0.987** |
| (Pavithra et al., 2023) | YOLOv8 | 50 especies de malezas (clases desbalanceadas). | 0.860 |
| (Iqbal et al., 2023) | DINO | 2 clases: maíz, malezas. | 0.758 |
| (Iqbal et al., 2023) | YOLOv5 large | 2 clases: maíz, malezas. | 0.854 |
| (Li & Shi, 2024) | DETR | 1 clase: brassica chinensis. | 0.779 |
| (Borgogno et al., 2024) | DETR | 12 clases: maíz, tomate y 10 especies de malezas. | 0.513 |
| El autor | YOLOv8 | 5 clases: papa, kikuyo, diente de león, lengua de vaca, otros. | 0.799 |
| Este trabajo | RT-DETR | 5 clases: papa, kikuyo, diente de león, lengua de vaca, otros. | 0.809 |

*Fuente:* Autoría propia.

El mejor resultado obtenido es del autor (Deng et al., 2024) con el modelo DINO, que alcanzó 0.959 mAP para detectar 8 tipos de malezas. Le siguen los modelos YOLOv8 y YOLOv5, que obtienen 0.860 mAP y 0.854 mAP, respectivamente. Este trabajo queda en cuarto lugar con 0.809 de mAP para la tarea de detección del cultivo de papas con tres tipos de malezas y la clase “otros” para especies de plantas no identificadas a simple vista como una categoría.

Centrándose en los modelos Transformer, el mejor modelo es DINO del autor Deng et al. (2024), que alcanza 0.959 mAP, seguido por el modelo RT-DETR del presente trabajo, que consigue 0.809 de mAP. Esto indica que hay una diferencia de 0.150 entre ambos modelos. No obstante, este comportamiento puede deberse al número abrumador de parámetros del modelo DINO, que es aproximadamente de 217.2 M de parámetros, frente a los 32 M de parámetros del modelo RT-DETR, lo que representa un aumento de casi siete veces, indicando que el modelo DINO tiene mayor capacidad de aprendizaje. Además, se debe considerar que en las 8 clases de malezas del dataset CottonWeedDet12 no se toma en cuenta las plantas no identificadas a simple vista, a diferencia de este trabajo, donde la clase "otros" penaliza bastante el rendimiento del modelo. Otro factor que influye es la altura a la que fue capturada las imágenes, dado que las imágenes del dataset CottonWeedDet12 son tomadas con cámaras portátiles (iPhone SE, iPhone 11 Pro, NIKON D3300, Canon EOS 4000D) a una altura de 0.4 metros (Dang et al., 2023), que es sumamente inferior a los 9-10 metros empleados en este trabajo. Al capturar imágenes a menor altura, se facilita el aprendizaje del modelo, dado que puede reconocer muchos más patrones de cada planta. También, es importante considerar que el modelo RT-DETR está preparada para tareas de detección en tiempo real a una tasa de 114 FPS, lo que es un cuello de botella real en los modelos DINO, dado que este modelo emplea aproximadamente 108.7 ms de inferencia por imagen (Dehaerne et al., 2022). Finalmente, el tamaño del modelo DINO es de aproximadamente 343 MB, en comparación de las 64 MB del modelo RT-DETR, es una arquitectura sumamente pesada que compensa su volumen con la precisión del modelo.

Con respecto a los demás modelos Transformer, el presente modelo es superior a los modelos propuestos por los autores Li & Shi (2024) con DETR (0.779 mAP), Iqbal et al. (2023) con DINO (0.758 mAP), Borgogno et al. (2024) con DETR (0.513 mAP) y Abuhani et al. (2023) con DETR (0.43 mAP). Por otra parte, el trabajo realizado por Iqbal et al. (2023) resulta contradictorio con el trabajo de Deng et al. (2024), puesto que ambos autores emplean el mismo modelo, el primero para detectar maíz y malezas (clasificación binaria), y el segundo para detectar 8 tipos de malezas. Esto se puede deberse a que, en el caso de la clasificación binaria, las distintas especies de malezas que conforman la clase “malezas” son muy dispersas y desiguales tanto en tamaño como en cantidad, lo que no ocurre con el dataset de 8 clases de malezas donde todas sus instancias son claras y uniformes.

Si se compara este trabajo con los modelos DETR, solo el modelo entrenado por el autor Li & Shi (2024), enfocado solo en la detección de coles, alcanza un rendimiento que puede igualar al modelo del presente trabajo. Sin embargo, los modelos de los autores Abuhani et al. (2023) y Borgogno et al. (2024) consiguen los peores puntajes. Esto se puede atribuir a ciertos factores. En el primer caso, se debe a que el modelo no terminó su entretenimiento debido a las 20 épocas aplicadas con un batch size de 2, como se muestra en la Tabla 5; estos parámetros influyen bastante para el aprendizaje de los modelos. En el segundo caso, puede deberse al desafiante dataset usado para la detección de maíz y tomate junto a 10 diferentes especies de malezas capturadas a 11 metros de altura (casi similar a la altura empleada en este trabajo), debido a estas características el modelo tiene dificultades en aprender esta amplia gama de patrones.

Este trabajo presenta similitudes con los resultados alcanzados por los modelos de la familia YOLO que fueron empleados para: la detección de 50 especies de malezas distintas, la clasificación de maíz y malezas, así como para la detección de papas y malezas. En el presente trabajo, se obtienen resultados entre 0.799 a 0.860 de mAP, esto indica que el modelo RT-DETR compite directamente con los modelos de última generación en tareas de detección en tiempo real.

A nivel global, el modelo RT-DETR del presente trabajo se encuentra situado en una buena posición dentro de los detectores Transformers competitivos, así como con los modelos YOLO. No obstante, existen restricciones como la limitada representación del dataset con solo cinco tipos de clases. También, es importante reconocer que en el presente trabajo no se manipuló directamente los componentes internos de la arquitectura (como los bloques AIFI, CCFF, etc.), dado que se adaptar mejor el modelo al modificar estos componentes. A pesar de ello, el modelo demostró ser lo suficientemente robusto para resolver el problema planteado.

# CONCLUSIONES

- La exploración de las investigaciones más recientes fue fundamental para el desarrollo del proyecto. Conocer las malezas comunes en campos de cultivos de papa/maíz y el proceso de control de las malezas fue clave a la hora de preparar del campo para el cultivo de malezas, así como para realizar los cuidados del mismo. Con respecto al modelo utilizado, se empleó la arquitectura RT-DETR sobre otros modelos Transformer, puesto que es un modelo de vanguardia que compite directamente con los modelos YOLO en tareas de detección en tiempo real y es considerado como el sucesor directo de la arquitectura DETR. La investigación sobre la metodología KDD, las herramientas de desarrollo y la norma ISO 25010 fue sustancial para llevar un marco metodológico sólido en el desarrollo del proyecto y, a la vez, ofrecer la calidad del software desarrollado. Finalmente, la exploración de los trabajos relacionados permitió conocer las metodologías empleadas y las limitaciones actuales que existen en la tarea de detección de malezas.
- El entrenamiento del modelo RT-DETR se basó principalmente en el ajuste de los hiperparámetros para las distintas versiones del dataset de malezas, que fueron dividas en distintas fases. Se llevó a cabo varias etapas experimentación que consistieron en cambios de optimizadores, ajuste de la tasa de aprendizaje, escalado de imagen, etc. La principal métrica empleada para la evaluación del rendimiento general del modelo fue mAP (Mean Average Precision), sin embargo, también se emplearon otras métricas como Precision, Recall, mAP@50:95. A partir del modelo baseline, se realizó las experimentaciones de forma iterativa y, mediante la comparativa de rendimiento con los modelos previos y el baseline, se consiguió definir el mejor modelo alcanzado. Para finalizar, el balanceo de clase aportó bastante en el rendimiento del modelo como se observa en la Tabla 31, donde se evidencia el rendimiento del dataset desbalanceado (versión 2) frente al dataset balanceado (versión 4), se obtiene una diferencia de aproximadamente 0.05 puntos.
- La aplicación web fue desarrollada con las tecnologías más modernas para garantizar el cumplimiento de las características de portabilidad y compatibilidad de la ISO 25010. La aplicación web fue alojada en la plataforma Microsoft Azure mediante la tecnología de Docker para el empaquetamiento de aplicaciones en contenedores y se empleó el mejor

modelo alcanzado para el despliegue. Con respecto al cumplimiento de la norma ISO 25010, se realizó la medición de portabilidad para evaluar la adaptabilidad (capacidad de visualización en diferentes equipos tecnológicos) y la facilidad de instalación/reinstalación, donde se obtuvo una puntuación general de 88.9% de portabilidad. También, se realizó la evaluación de compatibilidad, donde se consiguió el 100% en la subcaracterística de coexistencia. Esto indica que la aplicación web es altamente portable y coexiste sin problemas con las aplicaciones del día a día. Finalmente, se mantuvo una interfaz limpia y sencilla para facilitar el manejo de la aplicación para cualquier tipo de usuario.

- La evaluación del modelo RT-DETR se realizó mediante las métricas de rendimiento, la validación estadística y, finalmente, con la comparativa con los trabajos relacionados. Según la Tabla 33, el mejor modelo consiguió 0.8092 de mAP y 0.4482 de mAP@50:95 para la tarea de detección de cultivos y malezas en 5 diferentes categorías: cultivo de papa, diente de león, kikuyo, lengua de vaca y "otros" (malezas no identificadas en su propia clase) sobre el conjunto de datos de validación. Sin embargo, cuando se evalúa con el conjunto de datos de testeo, se consiguió 0.762 de mAP, es decir, 0.047 puntos de diferencia frente a los datos de validación. Esta discrepancia se debe principalmente a que los datos son completamente nuevos para el modelo, lo que afecta negativamente su rendimiento. De la misma forma, se demostró una alta correlación entre las métricas alcanzadas y el puntaje del valor p en las pruebas t-Student en cada categoría, no obstante, este nivel de concordancia se ve penalizado por la clase "otros", el cual obtiene un rendimiento muy bajo en la precisión media (Average Precision) y presenta diferencias significativas en las mediciones reales y predichas según la prueba t-Student. En cuanto a la comparativa con los trabajos relacionados, el presente modelo presenta resultados similares o superiores a las métricas conseguidas por otros autores en tareas similares, como se demuestra en la Tabla 49. A modo de conclusión, el modelo RT-DETR demostró un buen desempeño para la detección de cultivos y malezas, a pesar de la penalización de rendimiento de la clase "otros". El modelo generaliza bastante bien en datos nunca antes vistos, lo cual representa una contribución significativa en el campo de estudio.

## RECOMENDACIONES

- La investigación de la literatura más reciente es muy importante en el campo de la inteligencia artificial, puesto que, al ser una área muy cambiante, la información puede quedar obsoleta en poco tiempo. Por ello, es recomendable explorar trabajos que sean lo más recientes posibles y que no sobrepasen los 5 años de antigüedad. Esto permitirá al investigador tener un conocimiento más reciente y amplio de las estrategias y soluciones aplicadas para la resolución del problema, así como conocer cuáles son los retos actuales que se enfrenta en este campo. Por otra parte, también es importante las utilizar las últimas versiones de los Frameworks y librerías, en lo posible, puesto que, estas implementan mejoras de rendimiento y correcciones de errores de las versiones previas.
- Con respecto al dataset empleado, se recomienda ampliar las categorías plantas del presente trabajo con la finalidad de eliminar la clase “otros”, que representa un cuello de botella considerable para el aprendizaje de los modelos. Esta ampliación puede realizarse mediante la investigación de datasets de malezas públicos que cumplan con los parámetros de captura empleados en este trabajo. También, se puede optar por cultivar diferentes especies de malezas, ajustando la selección a los recursos propios. Con respecto al etiquetado de datos, se recomienda usar un equipo de personas especializadas en la anotación de imágenes con el propósito de acelerar el tiempo empleado en la preparación del dataset. Finalmente, es recomendable tener un dataset con suficientes datos (preferiblemente mayor a 1000 imágenes) y debidamente balanceado, que contenga una variedad considerable de especies capturadas en diversas etapas de crecimiento del cultivo para que el modelo generalice bien todos los distintos casos propuestos.
- La aplicación web demostró ser efectiva para la tarea de detección de malezas a partir de imágenes capturadas por drones. Sin embargo, para futuros trabajos se recomienda mejorar el aplicativo para la inferencia de malezas a partir del procesamiento de videos, lo cual permitiría la cuantificación completa de malezas en un campo de cultivos en un solo paso. Otra alternativa sería la implementación del modelo en un sistema embebido para la detección de malezas en tiempo real. La viabilidad de ambas opciones dependen estrictamente de los recursos de cómputo que se desee emplear. Para el primer caso se

puede emplear el modelo RT-DETR del presente trabajo sobre una máquina con capacidad de procesamiento gráfico (GPU) de 4 GB o más. En cambio, para integrar un sistema embebido para la detección de malezas en tiempo real, lo más recomendable sería optar por otra arquitectura más ligera como las versiones nano de YOLO e incorporarlo a una placa de Arduino o Raspberry Pi que soporten la ejecución de estos modelos.

- Para mejorar el rendimiento del modelo RT-DETR, además de la mejora del dataset, también es recomendable experimentar con diferentes modificaciones de la arquitectura, así como probar con diferentes combinaciones de hiperparámetros para adaptar el modelo a la resolución específica del problema planteado. Algunos componentes que se pueden modificar son los bloques AIFI y CCFM del codificador híbrido eficiente, así como las capas de decodificador y cabecera del modelo. Estas modificaciones se puede realizar con librerías que permitan un mayor grado de libertad de personalización, como Hugging Face, o directamente empleando el código fuente del modelo RT-DETR.

## REFERENCIAS


Abuhani, D. A., Hussain, M. H., Khan, J., Elmohandes, M., & Zualkernan, I. (2023). Crop and Weed Detection in Sunflower and Sugarbeet fields Using Single Shot Detectors. *2023 IEEE International Conference on Omni-Layer Intelligent Systems, COINS 2023*. https://doi.org/10.1109/COINS57856.2023.10189257

Arrieta, J. (2000). Manejo integrado de malezas en el cultivo de la papa. *Corporación Colombiana de Investigación Agropecuaria - AGROSAVIA*, 142–159. http://hdl.handle.net/20.500.12324/1637

Bàrberi, P., Abu-Irmaileh, B., Auld, B., Berti, A., Cardina, J., Chikoye, D., Elzein, A., Ferrero, A., Forcella, F., Kim, K.-U., Kroschel, J., Labrada, R., Jiménez, M. M., Madsen, K. H., Nagatalevu-Seniloli, M., Sattin, M., Shin, D.-H., Streibig, J. C., Teasdale, J. R., … Zaragoza, C. (2004). *Manejo de malezas para países en desarrollo Addendum I* (R. Labrada, Ed.). FAO Roma. https://www.fao.org/4/y5031s/y5031s0e.htm

Borgogno, E., Sarvia, F., De Petris, S., Orusa, T., Mesías-Ruiz, G. A., Peña, J. M., De Castro, A. I., Borra-Serrano, I., & Dorado, J. (2024). Cognitive Computing Advancements: Improving Precision Crop Protection through UAV Imagery for Targeted Weed Monitoring. *Remote Sensing 2024, Vol. 16, Page 3026*, *16*(16), 3026. https://doi.org/10.3390/RS16163026

Brownlee, J., Stefania, C., & Saeed, M. (2022). *Building Transformer Models with Attention: Implementing a Neural Machine Translator from Scratch in Keras* (Vol. 1). Machine Learning Mastery. https://books.google.com.ec/books?id=hNWaEAAAQBAJ

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-End Object Detection with Transformers. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12346 LNCS*, 213–229. https://doi.org/10.1007/978-3-030-58452-8_13

Chollet, F. and others. (2023). *Keras*. https://keras.io/

Dang, F., Chen, D., Lu, Y., & Li, Z. (2023). YOLOWeeds: A novel benchmark of YOLO object detectors for multi-class weed detection in cotton production systems. *Computers and Electronics in Agriculture*, *205*, 107655. https://doi.org/10.1016/J.COMPAG.2023.107655

Dehaerne, E., Dey, B., & Halder, S. (2022). A Comparative Study of Deep-Learning Object Detectors for Semiconductor Defect Detection. *ICECS 2022 - 29th IEEE International Conference on Electronics, Circuits and Systems, Proceedings*. https://doi.org/10.1109/ICECS202256217.2022.9971022

Deng, B., Lu, Y., & Xu, J. (2024). Weed database development: An updated survey of public weed datasets and cross-season weed detection adaptation. *Ecological Informatics*, *81*, 102546. https://doi.org/10.1016/J.ECOINF.2024.102546

DJI. (2024, April 30). *DJI Mavic Pro 2*. https://www.dji.com/global/mavic-2/info

Docker, Inc. (2023). *Docker Docs*. https://docs.docker.com/guides/walkthroughs/what-is-a-container/

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, *17*(3), 37–37. https://doi.org/10.1609/AIMAG.V17I3.1230

Gallagher, J. (2023, August 23). *What is Data Augmentation? The Ultimate Guide.* https://blog.roboflow.com/data-augmentation/

García, I. D. (2008). *Visión artificial y procesamiento digital de imágenes usando Matlab*. Cámara Ecuatoriana del Libro - Núcleo de Pichincha.

Guamangate, Y. K. M., Caisaluisa, J. L. M., & Cerda, V. del C. T. (2021). Measurement of usability and portability of a web application developed with PWA technology. *ConcienciaDigital*, *4*(4), 6–27. https://doi.org/10.33262/concienciadigital.v4i4.1882

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, *79*(8), 2554–2558. https://doi.org/10.1073/PNAS.79.8.2554

Iqbal, N., Manss, C., Scholz, C., Konig, D., Igelbrink, M., & Ruckelshausen, A. (2023). AI-Based Maize and Weeds Detection on the Edge with CornWeed Dataset. *Proceedings of the 18th Conference on Computer Science and Intelligence Systems, FedCSIS 2023*, 577–584. https://doi.org/10.15439/2023F2125

ISO/IEC 25023. (2016). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality*. https://inen.isolutions.iso.org/obp/ui/#iso:std:iso-iec:25023:ed-1:v1:en

Jocher, G. (2023, July 10). *Why is my initial learning rate value being ignored by YOLOv8? - Ultralytics - Issue #3629*. https://github.com/ultralytics/ultralytics/issues/3629#issuecomment-1630005867

Jocher, G. (2024, June 19). *Discussion #9536 - guides/hyperparameter-tuning/ - ultralytics*. https://github.com/orgs/ultralytics/discussions/9536#discussioncomment-9819803

Jocher, G., Knobloch, J., Noyce, M., Rizwan, M., & Akyon, F. C. (2024, October 1). *Train - Ultralytics YOLO Docs*. https://docs.ultralytics.com/modes/train/#train-settings

Jocher, G., Noyce, M., & Rizwan, M. (2023, November). *RT-DETR (Realtime Detection Transformer) - Ultralytics YOLO Docs*. https://docs.ultralytics.com/models/rtdetr/#pre-trained-models

Jocher, G., Noyce, M., Rizwan, M., & Mohammed, Y. (2024, October 1). *Hyperparameter Tuning - Ultralytics YOLO Docs*. https://docs.ultralytics.com/guides/hyperparameter-tuning/#what-metrics-should-i-use-to-evaluate-model-performance-during-hyperparameter-tuning-in-yolo

Jocher, G., & Rizwan, M. (2024, October 26). *Predict - Ultralytics YOLO Docs*. https://docs.ultralytics.com/modes/predict/#inference-arguments

Jocher, G., Rizwan, M., & Zhu, I. (2024, July 4). *Object Detection Datasets Overview - Ultralytics YOLO Docs*. Ultralytics Documentation. https://docs.ultralytics.com/datasets/detect/#ultralytics-yolo-format

Joint Technical Committee. (2011). *SISTEMAS E INGENIERÍA DE SOFTWARE — REQUISITOS Y EVALUACIÓN DE SISTEMAS Y CALIDAD DE SOFTWARE (SQuaRE) — MODELOS DE CALIDAD DEL SISTEMA Y SOFTWARE (ISO/IEC 25010:2011, IDT)* (Vol. 1). https://bit.ly/3RZXI48

Kaggle. (2024, April 30). *How To Use Kaggle*. https://www.kaggle.com/docs

Kavir, A., & Delgado, O. (2021). *Método para la estimación de maleza en cultivos de lechuga utilizando aprendizaje profundo e imágenes multiespectrales*. https://repositorio.unal.edu.co/handle/unal/80482

Li, H., & Shi, F. (2024). A DETR-like detector-based semi-supervised object detection method for Brassica Chinensis growth monitoring. *Computers and Electronics in Agriculture*, *219*, 108788. https://doi.org/10.1016/J.COMPAG.2024.108788

MAGAP. (2016). *La política agropecuaria ecuatoriana: hacia el desarrollo territorial rural sostenible: 2015-2025 I Parte* (Sector Público Gubernamental, Ed.; 1st ed.). https://bit.ly/46XOWtI

Meta Developers. (2022). *React*. https://react.dev/

Microsoft, V. S. C. (2024, October 29). *Developing inside a Container using Visual Studio Code Remote Development*. https://code.visualstudio.com/docs/devcontainers/containers

Naciones Unidas. (2016, January 1). *Objetivos de desarrollo sostenible*. https://www.un.org/sustainabledevelopment/es/sustainable-consumption-production/

Padilla, R., Netto, S. L., & Da Silva, E. A. B. (2020). A Survey on Performance Metrics for Object-Detection Algorithms. *International Conference on Systems, Signals, and Image Processing*, *2020-July*, 237–242. https://doi.org/10.1109/IWSSIP48289.2020.9145130

Pavithra, S., Kachroo, D., Kadam, V., Padala, H., & Purbey, R. (2023). Drone-Based Weed And Disease Detection In Agricultural Fields To Maximize Crop Health Using a Yolov8 Approach. *2023 IEEE 7th Conference on Information and Communication Technology, CICT 2023*. https://doi.org/10.1109/CICT59886.2023.10455507

Python Software Foundation. (2024). *Welcome to Python.org*. https://www.python.org/

Ronacher, A. (2023). *Flask Documentation*. https://flask.palletsprojects.com/en/2.3.x/

Rosebrock, A. (2019). *Deep Learning for Computer Vision with Python Starter Bundle* (3rd ed.). PyImageSearch.

Rothman, D. (2022). Transformers for natural language processing: build, train, and fine-tuning deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, and GPT-3. *Packt Publishing*, 564. https://www.packtpub.com/product/transformers-for-natural-language-processing-second-edition-second-edition/9781803247335

SENPLADES. (2021). *Plan de Creación de Oportunidades 2021-2025*. https://bit.ly/46VI5AX

Sensio, J. (2021, March 17). *SensIO*. https://juansensio.com/blog/062_multihead_attention

Simón, M., Golik, S., Constanza, M., & Campanela, C. (2018a). Maíz: Manejo de malezas. In *Cereales de verano* (pp. 133–150). Editorial de la Universidad Nacional de La Plata (EDULP). https://doi.org/https://doi.org/10.35537/10915/68613

Simón, M., Golik, S., Constanza, M., & Campanela, C. (2018b). Sorgo: Manejo de las malezas. In *Cereales de verano,* (pp. 284–292). Editorial de la Universidad Nacional de La Plata (EDULP). https://doi.org/https://doi.org/10.35537/10915/68613

Subramanian, S., Juarez, S., Breviu, C., Soshnikov, D., & Bornstein, A. (2024). *Learn the Basics — PyTorch Tutorial documentation*. https://pytorch.org/tutorials/beginner/basics/intro.html

Tensorflow Developers. (2023). *TensorFlow*. Zenodo. https://doi.org/https://doi.org/10.5281/zenodo.10126399

Thorp, K., & Tian, L. (2004). A review on remote sensing of weeds in agriculture. *Precision Agriculture*, *5*(5), 477–508. https://doi.org/10.1007/S11119-004-5321-1

Torres, A. (2018). Principales malezas del cultivo de la papa en la provincia de Petorca. *Ficha Técnica INIA La Cruz*, *37*. https://biblioteca.inia.cl/handle/20.500.14001/67106

Ultralytics. (2024, August 30). *Ultralytics YOLO Docs*. https://docs.ultralytics.com/modes/train/#why-choose-ultralytics-yolo-for-training

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, *2017-December*, 5999–6009. https://arxiv.org/abs/1706.03762v7

Vinueza, K. (2024). *Cuantificación automática de malezas utilizando imágenes adquiridas por un dron en campos de cultivos de maíz y/o papa mediante la red neuronal convolucional ResNet*. https://repositorio.utn.edu.ec/handle/123456789/15746

Zhao, Y., Lv, W., Xu, S., Wei, J., Wang, G., Dang, Q., Liu, Y., & Chen, J. (2024). DETRs Beat YOLOs on Real-time Object Detection. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 16965–16974. https://doi.org/10.1109/CVPR52733.2024.01605

# ANEXOS

**Anexo 1**

Imágenes capturadas sin anotación para el desarrollo del trabajo, enlace de Kaggle.

**Anexo 2**

Código fuente empleado en el presente trabajo, enlace del repositorio de GitHub.

**Anexo 3**

Cuatro versiones del dataset empleados en el presente trabajo, enlace de Kaggle.

**Anexo 4**

El mejor modelo RT-DETR alcanzado en el presente trabajo, enlace de Kaggle.

**Anexo 5**

Ejemplo de inferencia del malezas en tiempo real, enlace de Kaggle.

**Anexo 6**

Resultados de entrenamiento en formato JSON y Excel, enlace de Google Drive.

**Anexo 7**

Enlace de la aplicación web, https://detector-malezas.azurewebsites.net.