



DIVISIÓN DE ESTUDIOS POSGRADO E INVESTIGACIÓN

Opción 1.- TESIS

TRABAJO PROFESIONAL

**DESARROLLO DE UN SISTEMA CLASIFICADOR DE MADUREZ
EN CEREZAS DE CAFÉ MEDIANTE PROCESAMIENTO DE
IMÁGENES E INTELIGENCIA ARTIFICIAL**

**QUE PARA OBTENER EL GRADO DE:
MAESTRO EN INGENIERÍA
ELECTRÓNICA**

PRESENTA:

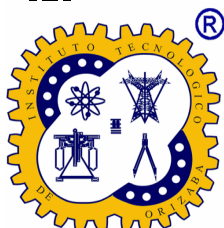
CARLOS ANTONIO MARTINEZ MIRANDA

DIRECTOR DE TESIS:

M.C. ENRIQUE ALEJANDRO CUÉLLAR CORTES

CODIRECTOR DE TESIS:

DR. OSCAR OSVALDO SANDOVAL GONZÁLEZ





Instituto Tecnológico de Orizaba
División de Estudios de Posgrado

Orizaba, Veracruz, **19/noviembre/2025**
Dependencia: **División de Estudios de
Posgrado e Investigación**
Asunto: **Autorización de Impresión**
OPCION: I

C. CARLOS ANTONIO MARTÍNEZ MIRANDA
Candidato(a) a Grado de Maestro(a) en:
INGENIERÍA ELECTRÓNICA
P R E S E N T E.-

De acuerdo con el Reglamento de Titulación vigente de los Centros e Institutos Tecnológicos Federales del Tecnológico Nacional de México, de la Secretaría de Educación Pública, y habiendo cumplido con todas las indicaciones que la Comisión Revisora le hizo respecto a su Trabajo Profesional titulado:

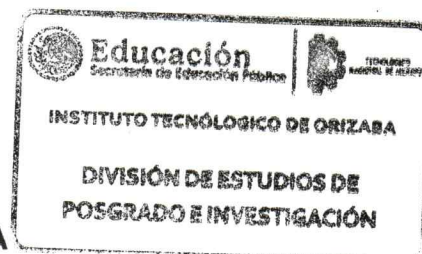
**“DESARROLLO DE UN SISTEMA CLASIFICADOR DE MADUREZ EN CEREZAS DE CAFÉ
MEDIANTE PROCESAMIENTO DE IMÁGENES E INTELIGENCIA ARTIFICIAL”**

Comunico a Usted que este Departamento concede su autorización para que proceda a la impresión del mismo.

A T E N T A M E N T E

Excelencia en Educación Tecnológica®
CIENCIA – TÉCNICA - CULTURA®

DRA. OFELIA LANDETA ESCAMILLA
JEFA DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN



OG-13-F06



2025
Año de
**La Mujer
Indígena**

Av. Oriente 9 #852 Colonia Emiliano Zapata, C.P. 94320
Orizaba, Veracruz. Tel. 01 (272) 1105360
e-mail: info@orizaba.tecnm.mx www.orizaba.tecnm.mx





Educación

Secretaría de Educación Pública



TECNOLÓGICO
NACIONAL DE MÉXICO



Instituto Tecnológico de Orizaba
División de Estudios de Posgrado

Orizaba, Veracruz, 18/noviembre/2025
Asunto: Revisión de trabajo escrito

C. OFELIA LANDETA ESCAMILLA
JEFA DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
PRESENTE.-

Los que suscriben, miembros del jurado, han realizado la revisión de la Tesis del (la) C.

CARLOS ANTONIO MARTÍNEZ MIRANDA

La cual lleva el título de:

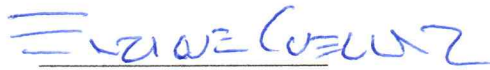
**“DESARROLLO DE UN SISTEMA CLASIFICADOR DE MADUREZ EN CEREZAS DE CAFÉ MEDIANTE
PROCESAMIENTO DE IMÁGENES E INTELIGENCIA ARTIFICIAL”**

Y concluyen que se acepta.

ATENTAMENTE

Excelencia en Educación Tecnológica®
CIENCIA – TÉCNICA - CULTURA®

PRESIDENTE: M.C. ENRIQUE ALEJANDRO CUELLAR
CORTÉS


FIRMA

SECRETARIO: DR. OSCAR OSVALDO SANDOVAL
GONZÁLEZ


FIRMA

VOCAL: DRA. BLANCA ESTELA GONZÁLEZ
SÁNCHEZ


FIRMA

VOCAL SUP.: DR. IGNACIO HERRERA ÁGUILAR


FIRMA

TA-09-18



2025
Año de
La Mujer
Indígena

Av. Oriente 9 #852 Colonia Emiliano Zapata, C.P. 94320
Orizaba, Veracruz. Tel. 01 (272) 1105360
e-mail: info@orizaba.tecnm.mx www.orizaba.tecnm.mx



CONTENIDO

AGRADECIMIENTOS	11
RESUMEN	12
ABSTRACT	14
INTRODUCCIÓN	16
PLANTEAMIENTO DEL PROBLEMA	17
JUSTIFICACIÓN	18
HIPÓTESIS	19
OBJETIVOS	20
OBJETIVOS ESPECIFICOS	20
CONTRIBUCIÓN AL CONOCIMIENTO	21
ALCANCES Y LIMITES	22
ALCANCES	22
LIMITES	22
CAPÍTULO 1: ANTECEDENTES	23
1.1 Cerezas de café	23
1.1.1 Anatomía de la cereza	23
1.1.2 Estado de maduración	24
1.2 Inteligencia artificial	25
1.2.1 Machine learning	26
1.2.2 Redes neuronales artificiales	27
1.2.2.1 Perceptrón	29

1.2.3	Deep learning	30
1.2.4	Redes neuronales convolucionales	30
1.2.4.1	Arquitectura de una CNN	31
1.2.5	YOLO	32
1.2.5.1	Mean Average Precision (mAP)	33
1.2.5.2	YOLO11	34
1.2.5.3	Características Principales	35
1.2.5.4	Tareas Soportadas	35
1.3	Entorno de desarrollo	36
1.3.1	Lenguaje de programación Python	36
1.3.2	Lenguaje de programación Dart	37
1.3.3	Anaconda	37
1.3.4	Jupyter Notebook	38
1.3.5	Google Colab	38
1.3.6	Visual Studio Code	39
1.3.7	Android Studio	40
1.3.8	Flutter	41
1.4	Servicio en la nube	44
1.4.1	Docker	44
1.4.2	Docker	44
1.4.3	Google Cloud	45
1.4.3.1	Google Cloud Run	46

1.4.4	Firestore	46
1.4.4.1	Authentication	47
1.4.4.2	Cloud Firestore	47
1.4.4.3	Firestore Storage	48
1.4.4.4	Cloud Functions	48
1.4.5	API Rest	49
CAPÍTULO 2: ESTADO DEL ARTE		51
CAPÍTULO 3: RED NEURONAL CONVOLUCIONAL PARA LA DETECCIÓN DE MADUREZ DE CEREZA DE CAFÉ		58
3.1	METODOLOGÍA	58
3.1.1	Selección de arquitectura	58
3.1.2	Recolección de datos.	60
3.1.3	Anotación de datos.	62
3.1.3.1	División de datos.	63
3.1.4	Preprocesamiento de datos.	65
3.1.4.1	Aumento de datos.	67
3.1.5	Entrenamiento y Evaluación.	69
3.2	RESULTADOS Y DISCUSIONES.	72
3.2.1	Análisis de resultados.	72
3.2.2	Ajuste fino.	77
3.2.3	Prueba del modelo.	82

CAPÍTULO 4: IMPLEMENTACIÓN DEL MODELO DE RED NEURONAL CONVOLUCIONAL.	88
4.1 BACKEND.	90
4.1.1 Contenerización.	90
4.1.1.1 Exportación del modelo	90
4.1.1.2 API REST.....	90
4.1.1.3 Docker	93
4.1.2 Despliegue en la nube.....	97
4.1.2.1 Google Cloud Run	97
4.1.2.2 Firebase.....	103
4.2 FRONTEND.	108
4.2.1 Aplicación móvil.....	108
CONCLUSIONES	122
RECOMENDACIONES.....	124
REFERENCIAS BIBLIOGRÁFICAS.....	125
PRODUCTOS ACADÉMICOS	136
ANEXOS	137
4.3 Anexo A : Código de la aplicación móvil.....	137
4.3.1 main.dart	137
4.3.2 navigation_bar_widget.dart	138
4.3.3 login_page.dart	140
4.3.4 register_page.dart	141

4.3.5	pref_usuarios.dart	142
4.3.6	auth.dart	143
4.3.7	storage_services.dart	144
4.3.8	Pantalla_00	146
4.3.9	camera_screen.dart	147
4.3.10	picture_screen.dart	148
4.3.11	Pantalla_01	150
4.3.12	mapa.dart	150
4.3.13	Pantalla_02	152
4.3.14	manage_place.dart	152
4.3.15	place.dart	154
4.3.16	place_draft.dart	155
4.3.17	place_dialog.dart	155
4.3.18	Pantalla_03	156
4.3.19	informacion.dart	157
4.4	Anexo B : Código de la API REST	159
4.4.1	main.py	159
4.4.2	app.py	166
4.5	Anexo C : Código de la función en Firebase Functions	171
4.5.1	main.py	171
4.6	Anexo D : Asesoría con Glenn Jocher, CEO de Ultralytics	179

LISTA DE FIGURAS

Figura 1.	Anatomía de la cereza de café. Fuente: Wintgens [7, Figura 1.1].	23
Figura 2.	Estados de madurez de la cereza de café: (a) Verde, (b) Semi-maduro, (c) Maduro, (d) Sobre-maduro.	25
Figura 3.	Anatomía de neurona cerebral humana. Fuente: Theobald [9, Figura 45].	27
Figura 4.	Los nodos, los pesos, y la función de activación de una red neuronal básica. Fuente: Theobald [9, Figura 46].	27
Figura 5.	Las tres capas general de una red neuronal. Fuente: Theobald [9, Figura 47].	28
Figura 6.	Representación visual de un perceptrón de una red neuronal. Fuente: Theobald [9, Figura 48].	29
Figura 7.	Reconocimiento facial usando Deep Learning. Fuente: Theobald [9, Figura 54].	30
Figura 8.	Arquitectura típica de una CNN. Fuente: Géron [10, Figura 14-12].	32
Figura 9.	Sistema de detección de objetos YOLO. Fuente: Redmon et al. [11, Figura 2].	33
Figura 10.	Intersección sobre Unión (IoU) para evaluación de cuadros delimitadores. Fuente: Terven et al. [12, Figura 3].	34
Figura 11.	Comparación de rendimiento de YOLO11 contra otros modelos. Fuente: Ultralytics [13, Figura 5]	35
Figura 12.	Logo de Python. Fuente: <i>Welcome to Python.org</i> [15].	36
Figura 13.	Logo de Dart. Fuente: <i>Dart overview</i> [17].	37
Figura 14.	Logo de Anaconda. Fuente: <i>Advance AI with Open Source</i> [18].	37
Figura 15.	Logo de Jupyter. Fuente: <i>The Jupyter Notebook — Jupyter Notebook 7.5.0b0 documentation</i> [19].	38
Figura 16.	Logo de Google Colab. Fuente: <i>Google Colab</i> [20].	39
Figura 17.	Logo de Visual Studio Code. Fuente: <i>Setting up Visual Studio Code</i> [21].	40
Figura 18.	Logo de Android Studio. Fuente: <i>Introducción a Android Studio</i> [22].	40
Figura 19.	Logo de Flutter. Fuente: <i>Flutter - Build apps for any screen</i> [23].	42
Figura 20.	Arquitectura de Flutter. Fuente: Moore [16].	43
Figura 21.	Logo de Docker. Fuente: <i>What is Docker?</i> [24].	44
Figura 22.	Logo de Google Cloud. Fuente: <i>Qué es Cloud Run</i> [26].	45
Figura 23.	Logo de Firebase. Fuente: <i>Documentación de Compilación Firebase Documentation</i> [27].	47

Figura 24. Estados de madurez de cerezas de café. Fuente: Tamayo-Monsalve et al. [29, Figura 1].	51
Figura 25. Detecciones de frutos de café realizadas por algoritmos YOLO considerando un tamaño de red de 800×800 píxeles para tres imágenes arbitrarias que representan frutos (A) en la sombra, (B) entre las hojas y (C) superpuestos. Fuente: Bazame et al. [30, Figura 3].	52
Figura 26. Pruebas en árbol. Fuente: Fuentes et al. [31, Figura 5].	53
Figura 27. Ejemplo del trabajo de muestra de la interfaz de la aplicación para teléfonos inteligentes. Fuente: Mimma et al. [5, Figura 18].	54
Figura 28. Cajas delimitadoras y etiquetas de clases generado por la red. Fuente: Madhavi et al. [32, Figura 18].	55
Figura 29. Metodología para la detección de madurez de cerezas de café	58
Figura 30. Tipos de entrenamiento en un modelo YOLO: (a) Training from Scratch y (b) Transfer Learning.	60
Figura 31. Estados de madurez de la cereza de café: (a) Verde, (b) Semi-maduro, (c) Maduro y (d) Sobre-maduro.	61
Figura 32. Captura de pantalla de la creación de un proyecto en Roboflow.	62
Figura 33. Proceso de etiquetado: (a) Captura de pantalla de la sección 'Annotate' en Roboflow y (b) Etiquetado de imágenes con cajas delimitadoras.	63
Figura 34. Capturas de pantalla del conjunto de datos: (a) Subconjunto de entrenamiento, (b) Subconjunto de validación y (c) Subconjunto de prueba.	64
Figura 35. Captura de pantalla de la versión del conjunto de datos en Roboflow: (a) Interfaz de Roboflow en la sección Dataset y (b) Interfaz de Roboflow en la sección Versions.	66
Figura 36. Preprocesamiento de imágenes: (a) Auto-Orient y (b) Resize.	67
Figura 37. Aumento de datos: (a) Imagen con Rotation, (b) Imagen con Brightness y (c) Imagen con Blur.	68
Figura 38. Captura de pantalla de la distribución del conjunto de datos en Roboflow.	69
Figura 39. Salidas visuales del modelo YOLO11m: (a) Matriz de confusión normalizada, (b) Curva F1-Confidence, (c) Curva Precision-Confidence, (d) Curva Precision-Recall, (e) Curva Recall-Confidence y (f) Resultados.	77

Figura 40.	Captura de pantalla de GPU seleccionada en Google Colab.	78
Figura 41.	Resultados gráficos del ajuste fino en el modelo YOLO11m: (a) Matriz de confusión normalizada, (b) curva F1-Confidence, (c) curva Precision-Confidence, (d) curva Precision-Recall, (e) curva Recall-Confidence y (f) Resultados.	82
Figura 42.	Resultados gráficos de la prueba del modelo YOLO11m: (a) Matriz de confusión normalizada, (b) curva F1-Confidence, (c) curva Precision-Confidence, (d) curva Precision-Recall y (e) curva Recall-Confidence.....	84
Figura 43.	Visualización de la predicción del modelo YOLO11m: (a) Predicción en categoría unripe, (b) Predicción en categoría semi-ripe, (c) Predicción en categoría ripe, (d) Predicción en categoría overripe y (e) Predicción del modelo en video.	87
Figura 44.	Diagrama de bloques de la implementación del sistema.	89
Figura 45.	Código de exportación del modelo YOLO11m en formato ONNX.	90
Figura 46.	Fragmento del código del archivo main.py.....	91
Figura 47.	Fragmento del código del archivo app.py.	93
Figura 48.	Código del Dockerfile utilizado para crear la imagen.....	94
Figura 49.	Imagen docker creada.	95
Figura 50.	Contenedor docker en ejecución.....	96
Figura 51.	Inferencia del modelo con el programa cURL: (a) Imagen de prueba, (b) Resultado de inferencia.	96
Figura 52.	Captura de pantalla de la creación de un nuevo proyecto en Google Cloud. .	97
Figura 53.	Visualización de repositorio en Artifact Registry.	98
Figura 54.	Salida de la terminal al crear el repositorio en Artifact Registry.....	99
Figura 55.	Visualización de los permisos en la sección IAM y administración en Google Cloud.....	100
Figura 56.	Visualización de la imagen Docker almacenada en el repositorio de Artifact Registry.....	101
Figura 57.	Visualización de la imagen Docker en función en Cloud Run.....	103
Figura 58.	Captura de pantalla de la creación de un nuevo proyecto en Firebase.....	104
Figura 59.	Visualización vinculación entre Firebase y Flutter.....	105
Figura 60.	Visualización de la habilitación de Firebase Storage.....	106
Figura 61.	Visualización del panel de Firebase Functions	107
Figura 62.	Captura de pantalla de las dependencias del proyecto: (a) dependencias del	

proyecto Flutter y (b) dependencias de desarrollo del proyecto Flutter.	109
Figura 63. Captura de pantalla de la configuración del archivo build.gradle.	110
Figura 64. Diagrama del flujo de trabajo de la aplicación móvil.	110
Figura 65. Pantallas para ingresar a la aplicación móvil: (a) pantalla de inicio de sesión y (b) pantalla para crear cuenta.....	111
Figura 66. Pantallas de la aplicación móvil: (a) pantalla inicial para tomar fotografía, (b) pantalla del mapa, (c) pantalla de galería, (d) pantalla de configuración.	112
Figura 67. Captura de pantalla al presionar botón de tomar fotografía: (a) antes de tomar fotografía, (b) después de tomar fotografía y (c) subiendo imagen a Storage.....	113
Figura 68. Resultados visualizados en la aplicación móvil: (a) información sobre las detecciones en la imagen y (b) ubicación de las imágenes en el mapa.	116
Figura 69. Resultados de la prueba en campo visualizados en la aplicación móvil: (a) captura de pantalla de la fotografía, (b) información sobre las detecciones en la fotografía, (c) captura de pantalla de la fotografía, (d) información sobre las detecciones en la fotografía.	120
Figura 70. Resultados de la prueba en campo visualizados en la aplicación móvil: (a) captura de pantalla de la fotografía, (b) información sobre las detecciones en la fotografía, (c) captura de pantalla de la fotografía, (d) información sobre las detecciones en la fotografía.	121
Figura 71. Capturas de pantalla de GitHub: (a) conversación con Glenn Jocher, CEO de Ultralytics y (b) perfil de Glenn Jocher en GitHub.....	180

LISTA DE TABLAS

Tabla 1.	Métricas de validación del modelo YOLO11n.....	73
Tabla 2.	Métricas de validación del modelo YOLO11m.....	73
Tabla 3.	Métricas de validación del modelo YOLO11x.....	74
Tabla 4.	Comparativa de métricas de resultados entre modelos YOLO11	75
Tabla 5.	Resultados del modelo YOLO11m en el conjunto de validación	85

AGRADECIMIENTOS

A mis padres y hermana, *Erendira Miranda Cerda, Carlos Antonio Martínez León y Monserrat Martínez Miranda, les agradezco profundamente su amor incondicional y su apoyo constante. Han sido mi pilar en momentos difíciles y mi motivación a seguir en el camino.*

A mi director y codirector de tesis, *M.C. Enrique Alejandro Cuéllar Cortés y Dr. Óscar Osvaldo Sandoval González, gracias por los consejos y apoyo que me permitieron finalizar este proyecto de investigación. Me han motivado a superar desafíos y en continuar en el mundo de la investigación.*

A mis compañeros y amigos, *les agradezco a todos mis amigos y compañeros dentro y fuera de la institución por su apoyo, gracias por todos los momentos que hemos vivido, su compañía en momentos de estrés y alegría han contribuido a que concluya esta bonita etapa.*

A la maestría en ingeniería electrónica y mis profesores, *les agradezco a todos los doctores y maestros de la maestría en ingeniería electrónica por enseñarme sus conocimientos y habilidades, gracias por darme la oportunidad de crecer como profesional y como persona.*

A TECNOM y SECIHTI, *Al Tecnológico Nacional de México y a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación, gracias por el apoyo institucional y los recursos proporcionados para el desarrollo de esta investigación.*

RESUMEN

La producción de café en México, uno de los principales en el mundo, se enfrenta a importantes retos debido a la falta de acceso tecnológico de los pequeños agricultores. La cosecha de las cerezas de café en su punto óptimo de madurez se realiza habitualmente con el criterio empírico, el cual influye directamente en la cantidad y calidad del producto final, ya que cada uno de los estados de madurez de la cereza del café poseen características físicas y químicas específicas.

La presente investigación, que tiene por título ‘Desarrollo de un sistema clasificador de madurez en cerezas de café mediante procesamiento de imágenes e inteligencia artificial’, propone el desarrollo de un sistema de clasificación de la madurez de las cerezas de café mediante el procesamiento de imágenes y la inteligencia artificial para pequeños productores de café. Dicho sistema basado en una red neural convolucional, específicamente YOLO11, tiene como objetivo reducir la dependencia al personal altamente cualificado mediante la identificación automática de las tonalidades de color de la cereza del café asociadas a sus diferentes etapas de madurez.

La metodología para realizar el sistema clasificador de madurez se basa en dos partes, el desarrollo de la red neuronal convolucional y la implementación de la misma:

- El desarrollo de la red neuronal convolucional se llevó a cabo mediante la adquisición, preprocesamiento y aumento de datos para posteriormente entrenar y validar la red neuronal convolucional, YOLO11, en 3 de sus tamaños y con esto, analizar y elegir el mejor modelo.
- La implementación del modelo de detección de la madurez utiliza el uso de la contenerización de una API REST en Docker, el despliegue en Google Cloud y una aplicación móvil para la visualización de las inferencias.

Los resultados del modelo YOLO11m demostró que fue el modelo más equilibrio, alcanzando un mAP@50 de 0.864 en las pruebas realizadas. La clase de cereza verde obtuvo el AP más alto (0.904), mientras que la clase de cereza sobre-madura presentó el más bajo (0.817), debido

al desequilibrio de la clase de cereza sobre-madura en el conjunto de datos, lo que genera que exista una mayor complejidad visual. Su implementación en una aplicación móvil se realizó de manera satisfactoria, teniendo el límite principal de la conexión a internet.

Esta investigación demostró la eficacia de utilizar técnicas de Deep Learning, en este caso la red neuronal convolucional, YOLO11, para tareas de cultivo de café como la clasificación de la madurez de cereza de café en condiciones reales. El sistema automatizado se presenta como una solución accesible con la posibilidad de optimizar la cosecha y mejorar la calidad del grano.

Palabras Clave: Café, Madurez, Clasificación Automática, Visión Computacional, YOLO11, Agricultura de Precisión, Docker, API REST, Google Cloud, Firebase, Flutter, Aplicación Móvil

ABSTRACT

Coffee production in Mexico, one of the world's leading producers, faces significant challenges due to small farmers' lack of access to technology. The harvesting of coffee cherries at their optimum point of ripeness is usually done using empirical criteria, which directly influences the quantity and quality of the final product, since each stage of ripeness of the coffee cherry has specific physical and chemical characteristics.

This research, entitled "Development of a coffee cherry ripeness classification system using image processing and artificial intelligence," proposes the development of a coffee cherry ripeness classification system using image processing and artificial intelligence for small coffee producers. This system, based on a convolutional neural network, specifically YOLO11, aims to reduce dependence on highly qualified personnel by automatically identifying the color tones of coffee cherries associated with their different stages of ripeness.

The methodology for implementing the ripeness classification system is based on two parts: the development of the convolutional neural network and its implementation:

- The convolutional neural network was developed by acquiring, preprocessing, and augmenting data to subsequently train and validate the convolutional neural network, YOLO11, in three of its sizes, thereby analyzing and selecting the best model.
- The implementation of the maturity detection model utilizes the containerization of a REST API in Docker, deployment on Google Cloud, and a mobile application for the visualization of inferences.

The results of the YOLO11m model showed that it was the most balanced model, achieving a mAP@50 of 0.864 in the tests performed. The green cherry class obtained the highest AP (0.904), while the overripe cherry class had the lowest (0.817), due to the imbalance of the overripe cherry class in the dataset, which generates greater visual complexity. Its implementation in a mobile application was successful, with the main limitation being the internet connection.

This research demonstrated the effectiveness of using deep learning techniques, in this case

the convolutional neural network YOLO11, for coffee cultivation tasks such as classifying coffee cherry ripeness in real conditions. The automated system is presented as an accessible solution with the potential to optimize harvesting and improve bean quality.

Keywords: Coffee, Maturity, Automatic Classification, Computer Vision, YOLO11, Precision Agriculture, Docker, REST API, Google Cloud, Firebase, Flutter, Mobile Application

INTRODUCCIÓN

Millones de personas consideran que la vida no sería la misma sin café, pues esta famosa bebida es imprescindible para empezar su día y realizar varias actividades como disfrutar un libro, ver una película o conversar con amigos que no han visto en un largo tiempo. En México, su producción rebasa el medio millón de toneladas al año [1], además de ubicarse entre los primeros lugares del ranking mundial de producción de café con países como Brasil, Colombia, Indonesia, Vietnam, entre otros [2].

Además, se mantiene como uno de los principales productores de América, ya que el país se mantiene con 600 mil hectáreas de cultivo distribuidas en 15 de los 32 estados de la república, en su mayoría por agricultores de pequeña escala de pueblos originarios [3]. Chiapas, Veracruz, Puebla, Oaxaca y Guerrero, con un 94 % de la producción nacional de café, son los encargados de proveer su consumo cotidiano en el país [3], y para obtener un producto de alta calidad, los caficultores deben de poner mucha atención en todas las etapas del proceso de la bebida. Una de estas etapas es la cosecha, donde se utiliza usualmente la experiencia humana, el cual influye directamente en la cantidad y calidad del café, ya que cada uno de los estados de madurez del fruto poseen características físicas y químicas específicas [4].

Con el fin de aplicar la tecnología en la etapa de cosecha de café que sea accesible para caficultores de producción pequeña, en esta investigación se propone la implementación de técnicas de procesamiento de imágenes y redes neuronales convolucionales en una aplicación móvil que permitirá la clasificación de la madurez de la cereza de café, tal como se muestra en investigaciones como Mimma et al. [5] y Muhlisin et al. [6] donde combinan el uso de redes neuronales con aplicaciones móviles para obtener herramientas digitales útiles en tareas relacionadas con el cultivo del café y otros frutos, promoviendo el uso de los avances tecnológicos para mejorar la sostenibilidad del sector.

PLANTEAMIENTO DEL PROBLEMA

Actualmente, la experiencia humana para tomar decisiones críticas en la etapa de la cosecha afecta considerablemente el potencial de los caficultores de pequeña escala. Esto debido a que el grado de maduración del fruto del café es uno de los factores más influyentes en el rendimiento del beneficio y en la calidad de la bebida [4], ya que los frutos con coloración verde disminuyen la calidad del café en todos los procesos postcosecha, produciendo bajos rendimientos, sabores y aromas rancios y características muy amargas; los frutos sobremaduros proporcionan sabores de tipo fermento y con alto porcentaje de daños por insectos; mientras que los frutos maduros, aquellos que exteriormente presentan una coloración rojiza uniforme, proporcionan la mejor calidad de bebida [4].

Por otra parte, el avance tecnológico enfocado a la cosecha del fruto de café para productores de café a pequeña escala es escaso, lo que requiere personal calificado que determine el momento adecuado para el corte de la cereza en los árboles de café y así lograr un producto que permita ser diferenciado y valorado con respecto a su competencia.

JUSTIFICACIÓN

La presente investigación se propone desempeñar un papel importante en el avance de los caficultores de pequeña escala a través del desarrollo de tecnología especializada. En la actualidad, estos productores se enfrentan a desafíos significativos debido a la falta de acceso a maquinaria y personal calificado para la determinación del momento óptimo del corte de la cereza en los arbustos de café.

La experiencia humana para tomar decisiones críticas en el proceso de cosecha afecta considerablemente el potencial de los caficultores de pequeña escala. Además, la falta de avance tecnológico específico para su escala de producción impide que puedan beneficiarse de herramientas accesibles y adaptadas a sus necesidades.

En este contexto, la propuesta de desarrollar un sistema clasificador de madurez en cerezas de café mediante procesamiento de imágenes e inteligencia artificial que permita determinar el momento óptimo para el corte de cereza en los arbustos de café busca abordar estas limitaciones de manera integral. Al disminuir la dependencia de personal altamente calificado, se pretende contribuir en la toma de decisiones para la cosecha. Esto no solo buscará que se tome las mejores decisiones posibles en la recolección, sino que también tiene el potencial de aumentar la calidad del café producido. Además, la implementación de esta tecnología puede tener un impacto significativo en la sostenibilidad durante la producción del café a pequeña escala, al proporcionar herramientas a los caficultores que les permitan tomar decisiones informadas para satisfacer las necesidades del presente sin comprometer el futuro de las siguientes generaciones y así garantizar el equilibrio entre el crecimiento económico, el cuidado del medio ambiente y el bienestar social.

HIPÓTESIS

Mediante la implementación de un sistema que utiliza procesamiento de imágenes e inteligencia artificial, se clasificarán las distintas tonalidades de color de las cerezas de café para determinar su madurez y así, indicar el momento correcto de corte de la cereza de café.

OBJETIVOS

Desarrollar e implementar un sistema clasificador de color de las cerezas de café para determinar su madurez mediante técnicas de procesamiento de imágenes e inteligencia artificial.

OBJETIVOS ESPECÍFICOS

- Entrenar la red neuronal convolucional para el proceso de clasificación del sistema.
- Realizar el despliegue de la red neuronal convolucional para su uso en el mundo real.
- Elaborar una interfaz gráfica para mostrar los resultados de la inferencia de la red neuronal convolucional.

CONTRIBUCIÓN AL CONOCIMIENTO

La presente investigación toma de base varios conocimientos que se mostrarán en los capítulos [ANTECEDENTES](#) y [ESTADO DEL ARTE](#) para desarrollar el sistema clasificador de madurez en cereza de café y a partir de esto, contribuye al conocimiento en el área de electrónica y visión computacional, ya que aborda el tema de la detección de objetos en un área no controlada, es decir, en un área donde existen variables imprevistas que pueden afectar al momento de la inferencia del modelo de red neuronal convolucional como en un plantío de árboles de café.

Dicho tema enfocado a la investigación actual, se ha reforzado gracias al conjunto de datos, el cual contiene una gran cantidad de imágenes donde la mayoría fueron tomadas directamente de plantíos de café, para obtener un mejor entrenamiento del modelo de red neuronal convolucional en la investigación actual y que también sea útil en futuros usos o para los interesados en proyectos con el tema relacionado.

También se contribuyó al mostrar los métodos utilizados para el despliegue del modelo de detección mediante la nube y la aplicación móvil, postulándose como una opción de herramienta durante la cosecha de café para los productores pequeños.

La principal aportación se centra en el modelo de detección de madurez de cereza de café junto con los resultados de sus predicciones obtenidos después de su entrenamiento que ayudan directamente al caficultor en la toma de decisiones en la etapa de la cosecha o a la comunidad investigadora para futuros proyectos.

ALCANCES Y LIMITES

Para poder proporcionarle dirección, claridad y coherencia a la presente investigación, a continuación, se describirán los límites y alcances del sistema.

ALCANCES

El sistema será capaz de:

- Detectar e identificar tonalidades de color para determinar la madurez de la cereza del café.

LIMITES

Para el funcionamiento del sistema se deberá:

- Considerar las características del dispositivo móvil, ya que la aplicación móvil debe de contar con buena cámara y solo soporta dispositivos Android con versiones 6.0 o superiores.
- Contar con una conexión a internet para vincularse con la base de datos.
- Tomar fotografías con buena iluminación y con zoom en la aplicación móvil.
- Contar con correo electrónico o cuenta de Gmail para ingresar a la aplicación móvil.

CAPÍTULO 1: ANTECEDENTES

En este apartado se describen los conceptos y tecnologías que se utilizaron para el desarrollo de la tesis para obtener una mejor comprensión del desarrollo de la investigación.

1.1 Cerezas de café

El fruto del cafeto es una drupa similar a una cereza, cuyos granos internos se utilizan para producir café tostado, soluble y licor de café Arcila-Pulgarín et al. [4]. Las principales especies cultivadas son *Coffea arabica* y *C. canephora* var. *robusta*, de acuerdo a Wintgens [7].

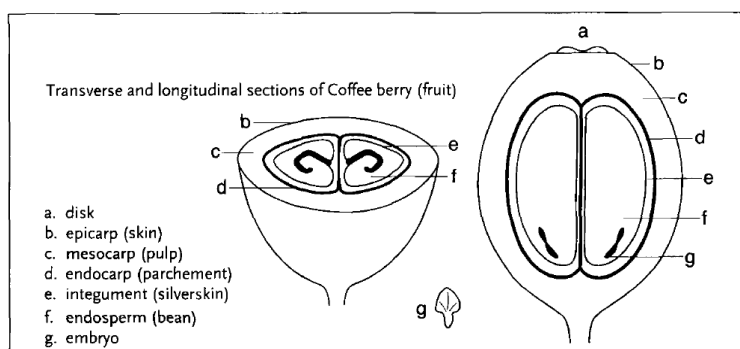


Figura 1: Anatomía de la cereza de café. Fuente: Wintgens [7, Figura 1.1].

1.1.1 Anatomía de la cereza

La estructura de la cereza (Figura 1) es la siguiente:

- **Epicarpio:** Capa externa (43.2 % del fruto) cuyo color varía según variedad y madurez
- **Mesocarpio:** Capa esponjosa rica en azúcares (11.8 % del fruto)
- **Endocarpio:** Membrana dura llamada pergamino (6.1 % del fruto)
- **Película plateada:** Membrana fina adherida al grano (0.2 % del fruto)
- **Semillas:** Endospermo o granos de café (38.9-55.4 % del fruto)

1.1.2 Estado de maduración

De acuerdo a la escala de maduración de la cereza de café que propone Marín-López et al. [8], se tomaron en cuenta los siguientes estados de maduración:

- Estado Inmaduro (Figura 2A): Contiene tonalidades verdes, las cuales se diferencian por poseer un pobre beneficio, bajos rendimientos y baja calidad de la bebida. Estos frutos se observan entre los 182 y 203 días después de la floración.
- Estado de Maduración Intermedia (Figura 2B): A partir de los 210 días después de la floración aproximadamente, se encuentran los frutos en estado semi-maduro. Estas cerezas poseen aceptables condiciones de beneficio y características para la bebida.
- Estado de Madurez Óptima (Figura 2C): Entre los 217 y 224 días después de la floración, se encuentran los frutos maduros, los cuales muestran condiciones óptimas de beneficio para la calidad de la bebida.
- Estado Sobremaduro (Figura 2D): De la misma manera, los frutos sobre-maduros se encuentran entre los 217 y 224 ddf, este estado presenta buena calidad de la bebida.

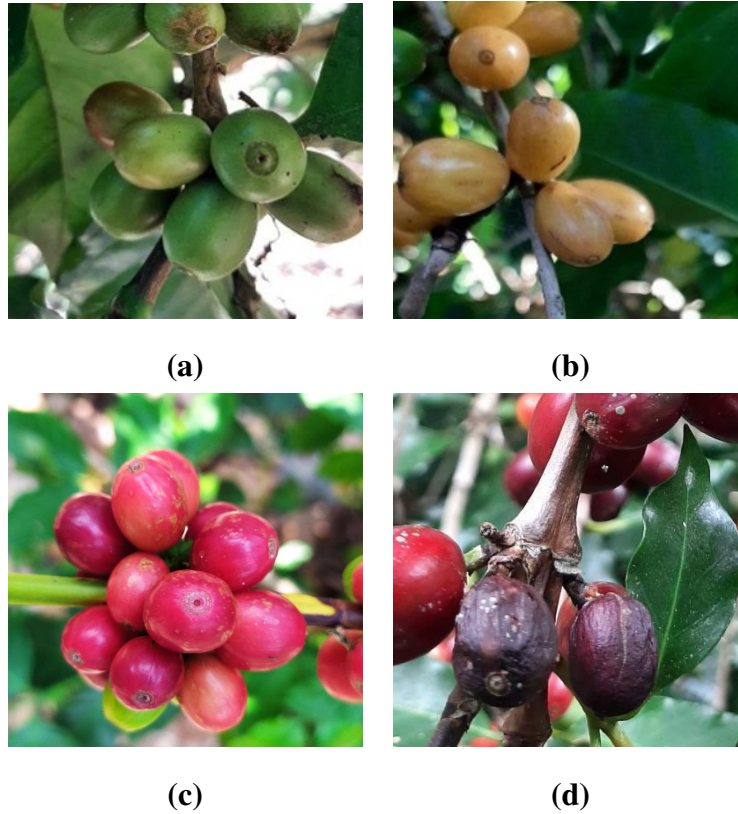


Figura 2: Estados de madurez de la cereza de café: (a) Verde, (b) Semi-maduro, (c) Maduro, (d) Sobre-maduro.

1.2 Inteligencia artificial

Como comenta Theobald [9] en su libro *Machine learning for absolute beginners*, la inteligencia artificial (IA) abarca la capacidad de las máquinas para realizar tareas inteligentes y cognitivas, similar a cómo la Revolución Industrial impulsó máquinas para tareas físicas.

La investigación en IA comenzó oficialmente en la Conferencia de Dartmouth de 1956, con gran optimismo inicial por avances en álgebra, aprendizaje de inglés y robótica. Sin embargo, en la década de 1970 surgió el ‘invierno de la IA’ debido a promesas exageradas y resultados limitados, con reducción de financiamiento.

Posteriormente, la IA renació mediante sistemas expertos con capacidades analíticas humanas, pero enfrentó un segundo invierno en los años 1990. El aprendizaje automático emergió como campo independiente utilizando probabilidad y estadística. El aumento de potencia

computacional permitió hitos como Deep Blue de IBM, que venció al campeón mundial de ajedrez en 1997.

Actualmente, la IA abarca múltiples campos como aprendizaje automático, visión artificial, procesamiento de lenguaje natural, planificación, sistemas expertos y robótica.

1.2.1 Machine learning

Como comenta Theobald [9] en su libro *Machine learning for absolute beginners*, a diferencia de la programación tradicional donde las decisiones están predefinidas, en el machine learning la computadora utiliza datos para construir modelos de decisión mediante razonamiento probabilístico y detección de patrones. El resultado depende del contenido de los datos, no de reglas preestablecidas.

En el machine learning, los datos se dividen en:

- **Datos de entrenamiento:** Reserva inicial para desarrollar el modelo.
- **Datos de prueba:** Para validar el rendimiento antes de implementación.

Los modelos se clasifican en:

- **Aprendizaje supervisado:** Utiliza datos etiquetados para extraer patrones de ejemplos conocidos.
- **Aprendizaje no supervisado:** Analiza relaciones entre variables sin etiquetas para descubrir patrones ocultos.
- **Aprendizaje semisupervisado:** Combina datos etiquetados y no etiquetados para mejorar la fiabilidad del modelo.
- **Aprendizaje por refuerzo:** El agente interactúa con el entorno, ejecuta acciones y recibe recompensas para aprender la política óptima, como comenta Géron [10].

1.2.2 Redes neuronales artificiales

Como menciona Theobald [9], las redes neuronales artificiales, son una técnica popular de machine learning para analizar datos a través de una red de capas de decisión, inspirándose en la estructura del cerebro humano.

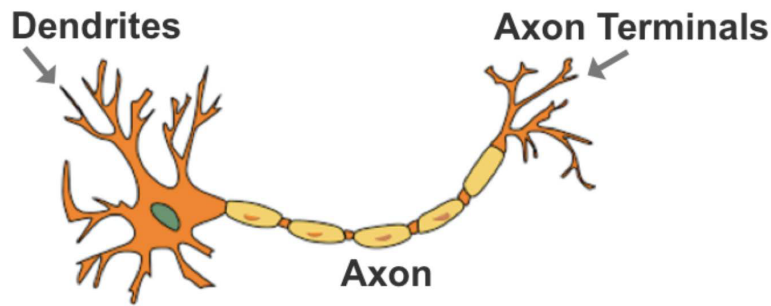


Figura 3: Anatomía de neurona cerebral humana. Fuente: Theobald [9, Figura 45].

El cerebro contiene neuronas interconectadas que reciben entradas mediante dendritas y emiten señales a través de axones, como se aprecia en la figura 3. De igual manera, las redes neuronales artificiales consisten en nodos de decisión interconectados que interactúan entre sí a través de bordes.

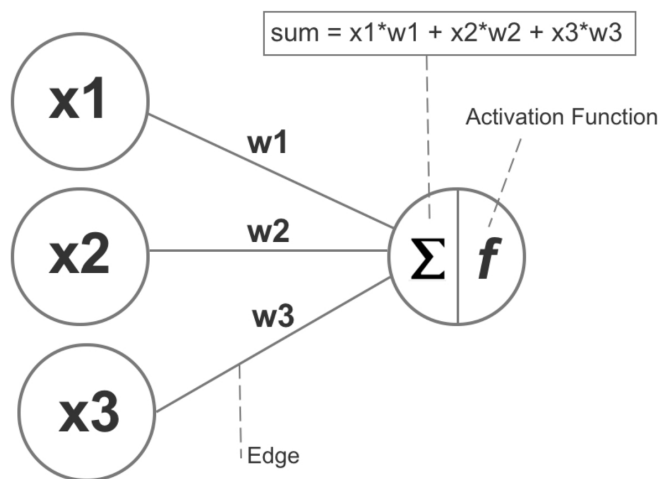


Figura 4: Los nodos, los pesos, y la función de activación de una red neuronal básica. Fuente: Theobald [9, Figura 46].

Los nodos de una red neuronal se dividen en capas y comienzan con una capa de entrada que

recibe datos sin procesar, como valores numéricos, texto, píxeles de imagen o sonido. Cada nodo de entrada envía información a la siguiente capa de nodos a través de los bordes de la red.

Como se muestra en la Figura 4, cada conexión en la red tiene un peso numérico ajustable según la experiencia. Si la suma de las conexiones satisface un umbral (función de activación), se activa un nodo en la siguiente capa; de lo contrario, no se activa, en un esquema de ‘todo o nada’. Los pesos únicos en cada conexión garantizan que los nodos se activen de manera diferenciada.

Mediante aprendizaje supervisado, se compara la salida predicha del modelo con la salida real conocida, midiendo la diferencia como ‘coste’. El entrenamiento busca minimizar este coste ajustando gradualmente los pesos de la red mediante el proceso de retropropagación, que opera en sentido inverso desde la capa de salida hacia la de entrada.

Una red neuronal típica se organiza en tres capas: entrada, oculta y salida, como se muestra en la Figura 5. La capa de entrada recibe los datos y detecta características, las capas ocultas procesan estas características, y la capa de salida presenta el resultado final.

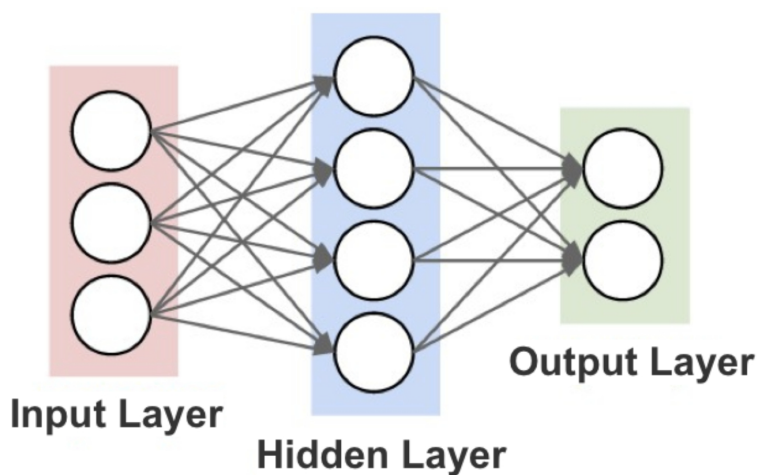


Figura 5: Las tres capas general de una red neuronal. Fuente: Theobald [9, Figura 47].

Las capas ocultas procesan información de manera encubierta, similar a cómo el cerebro humano reconoce patrones complejos de forma subconsciente. Al aumentar el número de capas ocultas, mejora la capacidad de analizar patrones complejos, lo que define al aprendizaje

profundo.

La forma más básica de red neuronal es el perceptrón, desarrollado en los años 1950 por Frank Rosenblatt, que utiliza propagación hacia adelante con flujo unidireccional de señales.

1.2.2.1 Perceptrón

Como menciona Theobald [9], el perceptrón fue diseñado como función de decisión que recibe entradas y produce una salida binaria. Su estructura básica consta de entradas, un procesador y una salida única, como se muestra en la Figura 6.

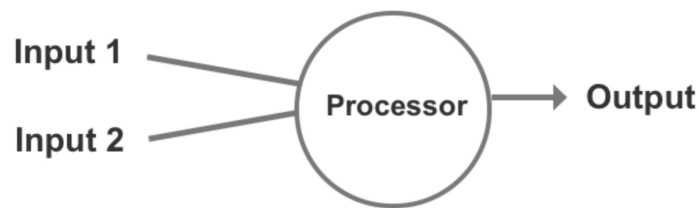


Figura 6: Representación visual de un perceptrón de una red neuronal. Fuente: Theobald [9, Figura 48].

Las entradas se procesan en la neurona, generando una salida binaria ('0' o '1'). La salida '1' activa la función de activación y pasa a la siguiente capa, mientras que '0' se ignora.

Como técnica de aprendizaje supervisado, el perceptrón sigue un proceso iterativo de cinco pasos:

1. Introducción de entradas al procesador
2. Aplicación de ponderaciones para estimar valores
3. Cálculo del error entre estimación y valor real
4. Ajuste de pesos según el error
5. Repetición hasta alcanzar la precisión deseada

1.2.3 Deep learning

Deep Learning es una subárea del Machine Learning que aborda patrones de datos complejos, como píxeles de imagen, donde los modelos superficiales dejan de ser fiables. Como menciona Theobald [9], las redes neuronales profundas interpretan numerosas características de entrada, descomponiendo patrones complejos en componentes más simples, como muestra la Figura 7.

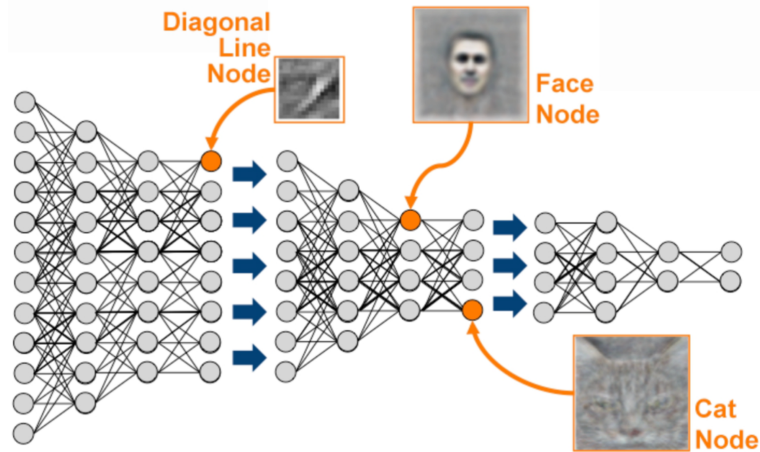


Figura 7: Reconocimiento facial usando Deep Learning. Fuente: Theobald [9, Figura 54].

Estas redes utilizan múltiples capas (5-10 como mínimo) para detectar características progresivamente más abstractas. El reconocimiento de objetos en vehículos autónomos, por ejemplo, puede emplear más de 150 capas. Otras aplicaciones incluyen análisis de series temporales, reconocimiento de voz y procesamiento de texto.

Los perceptrones multicapa han sido superados por técnicas más avanzadas como redes convolucionales, recurrentes, de creencias profundas y tensoriales recursivas. Las redes convolucionales destacan como las más populares y potentes, aunque los métodos continúan evolucionando.

1.2.4 Redes neuronales convolucionales

De acuerdo con Géron [10], las Redes Neuronales Convolucionales (CNN) son arquitecturas especializadas en el procesamiento de datos complejos, como imágenes. Surgieron del estudio

de la corteza visual y han evolucionado desde el neocognitrón (1980) hasta arquitecturas modernas como YOLO, logrando un rendimiento excepcional en algunas tareas visuales complejas como búsqueda de imágenes, vehículos autónomos, sistemas automáticos de clasificación de vídeo y otras tareas como el reconocimiento de voz y el procesamiento del lenguaje natural.

La arquitectura de una CNN consta principalmente de los siguientes componentes:

- **Capas Convolucionales:** Aplican filtros entrenables que detectan características locales mediante campos receptivos. Cada capa genera múltiples mapas de características que responden a patrones específicos (bordes, texturas, formas).
- **Funciones de Activación (ReLU):** Introducen no-linealidad después de cada capa convolucional, permitiendo modelar relaciones complejas.
- **Capas de Agrupamiento (Pooling):** Reducen dimensionalidad mediante submuestreo (generalmente máximo o promedio), proporcionando invariancia a traslaciones menores y reduciendo sobreajuste.
- **Capas Fully Connected:** En la parte final de la red, integran las características extraídas para generar predicciones finales.

1.2.4.1 Arquitectura de una CNN

La estructura característica de una CNN sigue un patrón jerárquico, como se muestra en la Figura 8, son las siguientes:

1. Bloques convolucionales + ReLU para extracción de características
2. Capas de pooling para reducción espacial
3. Repetición de este patrón con profundidad creciente
4. Capas fully connected para clasificación final

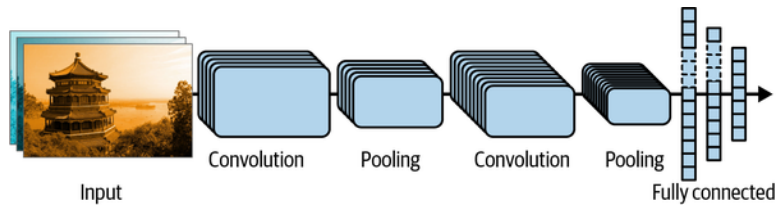


Figura 8: Arquitectura típica de una CNN. Fuente: Géron [10, Figura 14-12].

Esta organización permite que las primeras capas detecten patrones simples (bordes, colores) y las capas profundas combinen estos elementos en características complejas (objetos, formas).

Las variantes de esta arquitectura han logrado avances significativos, evidenciados en la reducción del error en el desafío ImageNet (ILSVRC) del 26 % al 2.3 % en seis años. Entre las arquitecturas más conocidas se encuentran LeNet-5, AlexNet, GoogLeNet, ResNet, SENet, Xception, EfficientNet y YOLO.

1.2.5 YOLO

De acuerdo con Géron [10], YOLO (You Only Look Once) es una arquitectura rápida y precisa para detección de objetos, propuesta por Redmon et al. [11], capaz de funcionar en tiempo real en video. Sus principales características incluyen:

- Considera objetos cuyo centro del cuadro delimitador se encuentra dentro de cada celda de la cuadrícula, como se aprecia en la figura 9.
- Coordenadas relativas a la celda, con (0,0) en esquina superior izquierda y (1,1) en inferior derecha.
- Genera dos cuadros delimitadores por celda para manejar objetos cercanos.
- Cada cuadro incluye su propia puntuación de objetividad.
- Produce distribución de probabilidad de clase por celda (20 clases en PASCAL VOC).

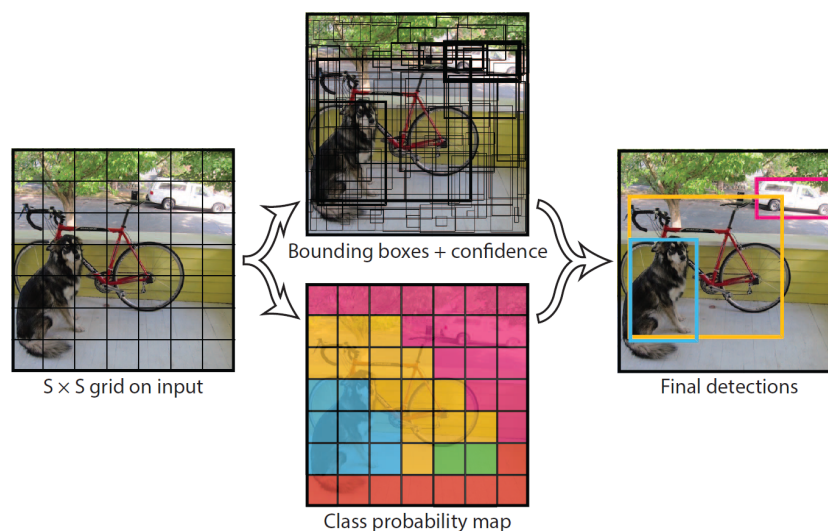


Figura 9: Sistema de detección de objetos YOLO. Fuente: Redmon et al. [11, Figura 2].

La asignación de clases se determina mediante correspondencia entre cuadros delimitadores y regiones dominantes en el mapa de probabilidad de clase.

YOLO que se desarrolló originalmente con Darknet, ha mejorado continuamente a lo largo de los años, con YOLOv2, YOLOv3 y YOLO9000, YOLOv4, YOLOv5, PP-YOLO, entre otros, donde cada versión trajo consigo mejoras impresionantes en velocidad y precisión, utilizando diversas técnicas.

1.2.5.1 Mean Average Precision (mAP)

El Mean Average Precision (mAP) es una métrica fundamental en detección de objetos que evalúa el equilibrio entre precisión y recall. Según Terven et al. [12], se basa en:

- **Precisión y Recall:** La precisión mide la exactitud de predicciones positivas, mientras el recall mide la proporción de positivos reales detectados. Existe un trade-off entre ambas métricas.
- **Curva Precisión-Recall:** Representa gráficamente esta relación para diferentes umbrales de confianza. El mAP calcula el área bajo esta curva.

- **Intersección sobre Unión (IoU):** Evalúa la calidad de los cuadros delimitadores midiendo la superposición entre predicciones y anotaciones reales (Figura 10).

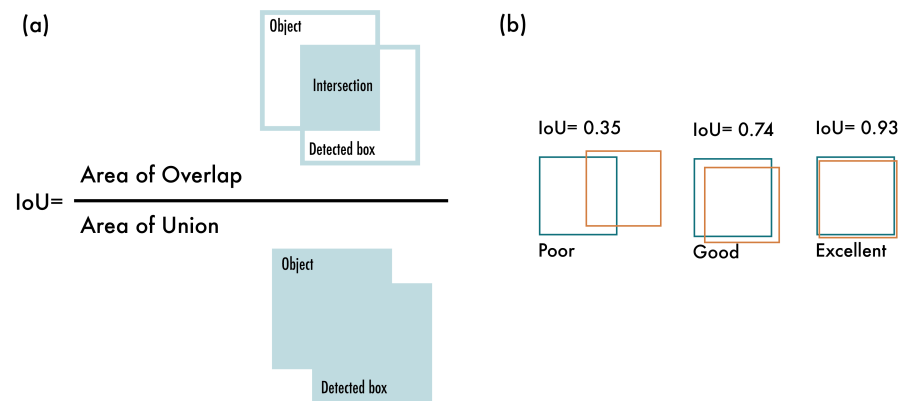


Figura 10: Intersección sobre Unión (IoU) para evaluación de cuadros delimitadores.

Fuente: Terven et al. [12, Figura 3].

El cálculo del mAP considera:

- **Umbral de IoU:** Se define un umbral mínimo (ej. 0.5) para considerar una predicción como correcta (mAP@0.5)
- **Múltiples categorías:** Se calcula AP para cada clase y se promedia (mAP)
- **Múltiples umbrales:** En COCO se usa mAP@[.50:.95] promediando umbrales de IoU

Esta métrica proporciona una evaluación comprehensiva del rendimiento en detección de objetos, considerando tanto la clasificación como la localización espacial.

1.2.5.2 YOLO11

De acuerdo a la documentación de Ultralytics Ultralytics [13], YOLO11 es la última iteración de la serie YOLO de Ultralytics, que establece nuevos estándares en precisión, velocidad y eficiencia para detectores de objetos en tiempo real. Como muestra la Figura 11, supera a modelos anteriores con mejoras significativas en arquitectura y métodos de entrenamiento.

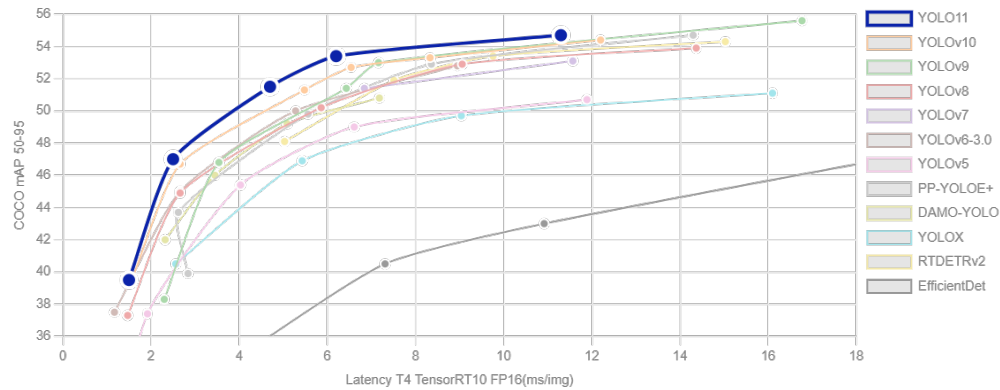


Figura 11: Comparación de rendimiento de YOLO11 contra otros modelos. Fuente: Ultralytics [13, Figura 5]

1.2.5.3 Características Principales

- **Eficiencia optimizada:** Diseños refinados que balancean velocidad y rendimiento
- **Mayor precisión con menos parámetros:** YOLO11m logra mayor mAP en COCO usando 22 % menos parámetros que YOLOv8m
- **Adaptabilidad multiplataforma:** Compatible con dispositivos edge, cloud y GPUs NVIDIA
- **Amplio rango de tareas:** Soporte para múltiples aplicaciones de visión artificial

1.2.5.4 Tareas Soportadas

- Detección de objetos y segmentación de instancias
- Clasificación de imágenes y estimación de pose
- Detección de objetos orientados (OBB)

1.3 Entorno de desarrollo

1.3.1 Lenguaje de programación Python

De acuerdo al libro *Image Processing with Python* de Ansari y Bajaj [14], Python (véase la figura 12) es un lenguaje de programación robusto, popular, orientado a objetos, interpretado y de alto nivel, adaptable para procesos tanto sencillos como complejos. Es compatible con diversos lenguajes y ampliamente utilizado como backend.



Figura 12: Logo de Python. Fuente: *Welcome to Python.org* [15].

Python se distingue por características prácticas esenciales:

1. **Fácil de aprender y usar:** Ideal para principiantes, permite codificar algoritmos avanzados de IA, machine learning y visualización de datos.
2. **Lenguaje multiplataforma:** Portable, se instala en Windows, Linux, UNIX, macOS.
3. **Gratuito y de código abierto:** Accesible para descarga desde su sitio oficial.
4. **Amplia biblioteca estándar:** Numerosas bibliotecas listas para usar (OpenCV, Scikit-image, Matplotlib) en diversos campos.

Entre sus aplicaciones destacan:

- Aplicaciones de video, audio y preprocesamiento de imágenes.
- Desarrollo web, juegos y aplicaciones de negocios.
- Ciencia de datos, visualización, Machine Learning e Inteligencia Artificial.

1.3.2 Lenguaje de programación Dart

De acuerdo con Moore [16] en el libro *Mastering Flutter*, Flutter utiliza el lenguaje Dart (véase la figura 13), desarrollado por Google. Originalmente creado para la web, es similar a JavaScript en sintaxis (variables con var, palabras clave async/await). Google ha mejorado continuamente el lenguaje, especialmente con seguridad nula en la versión 2.0 para prevenir fallos accidentales.



Figura 13: Logo de Dart. Fuente: *Dart overview* [17].

1.3.3 Anaconda

En la documentación oficial de *Advance AI with Open Source* [18], se define a Anaconda (véase la figura 14) como una plataforma integral para paquetes de Python que resuelve problemas de gestión de software de código abierto.



Figura 14: Logo de Anaconda. Fuente: *Advance AI with Open Source* [18].

Los proyectos de código abierto en Python dependen de múltiples dependencias gestionadas independientemente por mantenedores globales. Esto requiere una herramienta que administre paquetes, reconozca arquitecturas de hardware y mantenga proyectos separados para garantizar funcionalidad.

Conda es el administrador de paquetes, entornos y dependencias de Anaconda. Es multiplataforma (Windows, macOS, Linux), multilingüe (Python, R, C/C++, Rust, Go) y garantiza compatibilidad de paquetes. Sin un administrador como Conda, las cadenas de dependencias pueden volverse inmanejables con versiones incompatibles.

1.3.4 Jupyter Notebook

La documentación oficial de *The Jupyter Notebook — Jupyter Notebook 7.5.0b0 documentation* [19] define a Jupyter Notebook (véase la figura 15) como una aplicación para creación de cuadernos computacionales bajo el Proyecto Jupyter, que ofrece formas interactivas de crear prototipos, explicar código, explorar datos y compartir ideas.



Figura 15: Logo de Jupyter. Fuente: *The Jupyter Notebook — Jupyter Notebook 7.5.0b0 documentation* [19].

Los cuadernos extienden la computación interactiva mediante una aplicación web que captura todo el proceso de cálculo: desarrollo, documentación, ejecución de código y comunicación de resultados. Jupyter Notebook combina:

- **Aplicación web:** Editor basado en navegador para creación interactiva de cuadernos.
- **Documentos computacionales:** Documentos compartibles que combinan código, descripciones, datos, visualizaciones y controles interactivos.

1.3.5 Google Colab

En la documentación oficial de *Google Colab* [20], se define a Google Colab (véase la figura 16) como un servicio alojado de notebooks Jupyter que no requiere instalación y proporciona acceso gratuito a recursos computacionales, incluyendo GPUs y TPUs. Es ideal para aprendizaje automático, ciencia de datos y educación.



Figura 16: Logo de Google Colab. Fuente: *Google Colab* [20].

Colab permite escribir y ejecutar código Python en el navegador con:

- Sin configuración requerida.
- Acceso gratuito a GPUs.
- Facilidad para compartir documentos.

Los notebooks de Colab son entornos interactivos que combinan código ejecutable y texto enriquecido en un único documento, incluyendo imágenes, HTML, LaTeX y más. Los documentos se almacenan en Google Drive y pueden compartirse fácilmente para colaboración.

1.3.6 Visual Studio Code

De acuerdo con *Setting up Visual Studio Code* [21], Visual Studio Code (véase la figura 17) es un editor de código optimizado con soporte para operaciones de desarrollo como depuración, ejecución de tareas y control de versiones. Su objetivo es proporcionar solo las herramientas que un desarrollador necesita para un ciclo rápido de creación y depuración de código y deja flujos de trabajo más complejos en manos de IDE con más funciones, como IDE de Visual Studio.

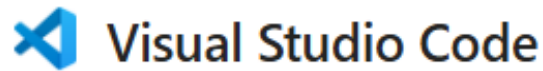


Figura 17: Logo de Visual Studio Code. Fuente: *Setting up Visual Studio Code* [21].

1.3.7 Android Studio

Según la documentación oficial de *Introducción a Android Studio* [22], Android Studio (véase la figura 18) es el entorno de desarrollo integrado (IDE) oficial para desarrollo de aplicaciones Android, basado en IntelliJ IDEA.



Figura 18: Logo de Android Studio. Fuente: *Introducción a Android Studio* [22].

Sus características principales son:

- Sistema de compilación flexible basado en Gradle.
- Emulador rápido y funcional.
- Desarrollo unificado para todos los dispositivos Android.
- Ediciones en tiempo real para actualizaciones inmediatas.
- Integración con GitHub y plantillas de código.
- Herramientas de testing y análisis Lint.
- Soporte para C++ y NDK.

Cada proyecto incluye módulos con archivos de código y recursos:

- Módulos de apps Android.
- Módulos de biblioteca.
- Módulos de Google App Engine.

La vista de proyecto organiza:

- **manifests:** Archivo AndroidManifest.xml.
- **java:** Código fuente Kotlin/Java y tests JUnit.
- **res:** Recursos no código (cadenas UI, imágenes).

Android Studio utiliza Gradle con el complemento Android para:

- Personalizar y extender el proceso de compilación.
- Crear múltiples APKs desde el mismo proyecto.
- Reutilizar código y recursos entre conjuntos de fuentes.

Los archivos de compilación (build.gradle.kts para Kotlin, build.gradle para Groovy) configuran automáticamente los proyectos importados.

1.3.8 Flutter

De acuerdo con Moore [16], Flutter (véase la figura 19) es un kit de herramientas de interfaz de usuario de Google para móviles, ordenadores y la web. Utiliza una IU declarativa basada en widgets, donde todo es un widget, pudiéndose usar los existentes o crear unos propios. Funciona en múltiples plataformas con prácticamente el mismo código, requiriéndose código adicional solo para funcionalidades específicas.



Figura 19: Logo de Flutter. Fuente: *Flutter - Build apps for any screen* [23].

Presentado en 2015 como “Sky” para Android usando Dart, Flutter 1.0 se lanzó en 2018. La versión 2 (2021) introdujo un motor web y seguridad nula en Dart. Flutter 2.5 (2021) añadió soporte para Material Design y Flutter 3 (2022) compatibilidad total con escritorios e interoperabilidad con iOS.

La arquitectura de Flutter tiene varias capas, como se muestra en la figura 20:

- **Framework** (Capa superior en Dart): Incluye widgets, y las bibliotecas Material y Cupertino.
- **Engine** (C/C++): No requiere uso directo.
- **Embedder**: Específica para cada plataforma.

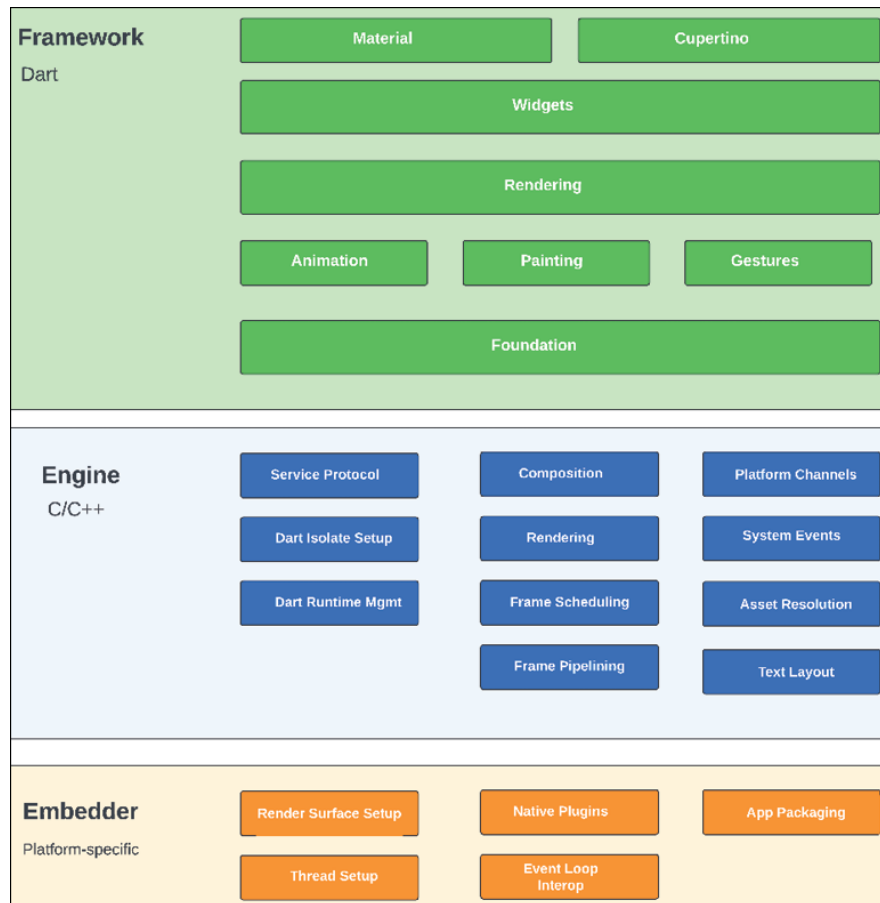


Figura 20: Arquitectura de Flutter. Fuente: Moore [16].

Las aplicaciones se compilan a código nativo para un rendimiento óptimo. Originalmente usaba Skia para gráficos, migrando ahora a Impeller. Para extender funcionalidades, se pueden usar paquetes (Dart) o plugins (código nativo por plataforma).

Sus ventajas incluyen:

- **Desarrollo rápido:** Con una IU declarativa y la función de recarga en caliente para actualizar la aplicación al instante.
- **Rendimiento nativo** al compilarse a código nativo.
- **Código abierto.**
- **Compilación para cualquier plataforma** con una sola base de código, agilizando el desarrollo y reduciendo la necesidad de personal.

1.4 Servicio en la nube

1.4.1 Docker

1.4.2 Docker

Según *What is Docker?* [24], Docker (véase la figura 21) es una plataforma abierta para desarrollar, distribuir y ejecutar aplicaciones. Permite separar las aplicaciones de la infraestructura, facilitando la entrega rápida de software y gestionando la infraestructura de la misma manera que las aplicaciones. Esto reduce significativamente el tiempo entre la escritura del código y su ejecución en producción.



Figura 21: Logo de Docker. Fuente: *What is Docker?* [24].

Docker utiliza contenedores, entornos ligeros y aislados que contienen todo lo necesario para ejecutar una aplicación, eliminando dependencias del sistema host. Permiten ejecutar múltiples contenedores simultáneamente y garantizan consistencia entre diferentes entornos.

Docker proporciona herramientas para gestionar el ciclo de vida de los contenedores:

- Desarrollo de aplicaciones usando contenedores.
- Distribución y pruebas con el contenedor como unidad.
- Implementación en producción, ya sea en centros de datos o en la nube.

La plataforma ofrece alta portabilidad, ejecutándose en laptops, máquinas virtuales, centros de datos o nubes. Facilita la gestión dinámica de cargas de trabajo, permitiendo escalar aplicaciones según necesidades.

Los principales componentes de Docker son:

- **Imágenes:** Plantillas de solo lectura con instrucciones para crear contenedores. Se construyen mediante Dockerfiles, donde cada instrucción crea una capa. Solo se reconstruyen las capas modificadas, esto es parte de lo que hace que las imágenes sean tan ligeras y eficientes.
- **Contenedores:** Instancias ejecutables de imágenes. Pueden crearse, iniciarse, detenerse o eliminarse mediante API o CLI. Están aislados por defecto, pero pueden conectarse a redes y almacenamiento. Al eliminar un contenedor, desaparecen todos los cambios en su estado que no se almacenen en el almacenamiento persistente.

1.4.3 Google Cloud

En el libro *Building serverless applications with Google Cloud Run* de Venema [25], se menciona que Google Cloud (véase la figura 22), consiste en recursos físicos y virtuales (como máquinas virtuales) distribuidos globalmente en regiones y zonas. Cada región contiene zonas aisladas entre sí, identificadas con nombres combinados (ej: asia-east1-a). Esta distribución ofrece ventajas de redundancia y menor latencia, aunque impone reglas sobre el uso conjunto de recursos.



Figura 22: Logo de Google Cloud. Fuente: *Qué es Cloud Run* [26].

Google Cloud inició en 2008 con App Engine, una plataforma sin servidor pionera pero con limitaciones que la hacían adecuada principalmente para proyectos nuevos. Casos exitosos incluyen Snapchat y Spotify, aunque tuvo penetración limitada.

En 2012, respondiendo a la demanda del mercado, Google lanzó Compute Engine. La mentalidad de Google siempre ha estado orientada hacia infraestructuras sin servidor, basándose en Borg, su sistema de contenedores a escala planetaria desarrollado desde al menos 2005.

Borg permite a los desarrolladores enfocarse en aplicaciones en lugar de detalles operativos, logrando hiperescalabilidad.

1.4.3.1 Google Cloud Run

Venema [25] comenta que Cloud Run es una plataforma de aplicaciones totalmente gestionada que ejecuta contenedores invocables mediante solicitudes o eventos. Abstrayendo toda la gestión de infraestructura, permite a los desarrolladores concentrarse en crear aplicaciones.

Soporta cualquier lenguaje de programación mediante imágenes de contenedor, y ofrece despliegue automático para lenguajes como Go, Node.js, Python, Java, .NET y Ruby.

Cloud Run está diseñado para integrarse bien con otros servicios de Google Cloud, permitiendo crear aplicaciones con todas las funciones sin necesidad de gestionar clústers o infraestructura.

Los recursos pueden ejecutarse como:

- **Servicios:** Proporcionan infraestructura para endpoints HTTPS confiables, requiriendo que el código escuche en un puerto TCP y maneje solicitudes HTTP.
- Trabajos.
- Grupos de trabajadores.

Todos comparten el mismo entorno y pueden usar las mismas integraciones con servicios de Google Cloud.

1.4.4 Firebase

Según Moore [16] y la documentación oficial de *Documentación de Compilación | Firebase Documentation* [27], Firebase (véase la figura 23) es un conjunto de servicios en línea adquirido por Google en 2014. Comenzó con Firebase Realtime Database, expandiéndose posteriormente con servicios como Hosting, Authentication, Analytics, Crashlytics y Cloud Firestore. Muchos desarrolladores utilizan Firebase para almacenar datos, gestionar notificaciones, controlar fallos, distribuir aplicaciones y configuración remota.



Figura 23: Logo de Firebase. Fuente: *Documentación de Compilación | Firebase Documentation* [27].

1.4.4.1 Authentication

Firebase Authentication proporciona servicios backend, SDK fáciles de usar y bibliotecas de UI para autenticar usuarios. Soporta múltiples métodos:

- Contraseñas y números de teléfono.
- Proveedores federados (Google, Facebook, Twitter).
- Estándares OAuth 2.0 y OpenID Connect.

Identity Platform ofrece funciones adicionales como autenticación multifactor, registros de auditoría y soporte SAML. El flujo de autenticación implica obtener credenciales del usuario, verificarlas con los servicios backend y gestionar el acceso a datos basado en tokens.

1.4.4.2 Cloud Firestore

Cloud Firestore es una base de datos documental NoSQL que almacena datos en documentos organizados en colecciones. Se diferencia de Realtime Database en:

- **Realtime Database:**
 - Almacena datos como árbol JSON único.
 - Consultas recuperan sub-árbol completo.
 - Optimizada para sincronización en tiempo real.
 - Mejor para conjuntos de datos pequeños.

■ **Cloud Firestore:**

- Almacena datos en documentos y colecciones.
- Consultas más potentes y flexibles.
- Mejor para conjuntos de datos grandes.
- Modelo de datos jerárquico escalable.

Ambas ofrecen soporte sin conexión y seguridad basada en reglas, pero difieren en modelos de precios.

1.4.4.3 Firebase Storage

Cloud Storage for Firebase es un servicio de almacenamiento de objetos construido sobre Google Cloud Storage:

- Almacena contenido generado por usuarios (fotos, videos).
- Maneja automáticamente conexiones de red deficientes.
- Se integra con Firebase Authentication para control de acceso.
- Escala automáticamente sin necesidad de migración.

1.4.4.4 Cloud Functions

Cloud Functions para Firebase es un framework sin servidores que ejecuta código backend en respuesta a eventos:

- Soporta JavaScript, TypeScript y Python.
- No requiere gestión de servidores.
- Se escala automáticamente según la carga.
- Se puede activar mediante:

- Solicitudes HTTPS.
- Eventos de Firebase.
- Admin SDK.
- Cloud Scheduler.

1.4.5 API Rest

De acuerdo a Kleppmann [28] en su libro *Designing data-intensive applications*, comenta que la comunicación en red se organiza comúnmente mediante clientes y servidores. Los servidores exponen una API (servicio) a través de la red, y los clientes se conectan para realizar solicitudes a esa API.

En la web, los clientes (navegadores) realizan solicitudes a servidores web:

- Solicitudes GET para descargar recursos (HTML, CSS, JavaScript, imágenes).
- Solicitudes POST para enviar datos al servidor.

La API utiliza protocolos y formatos estandarizados (HTTP, URL, SSL/TLS, HTML), permitiendo la interoperabilidad entre cualquier navegador y sitio web.

Cuando se utiliza HTTP como protocolo subyacente, se denomina **servicio web**. Existen dos enfoques principales:

- **REST**: Filosofía de diseño basada en principios HTTP que enfatiza:
 - Formatos de datos sencillos.
 - URL para identificar recursos.
 - Funciones HTTP para caché, autenticación y negociación de contenido.
- **SOAP**: Enfoque diametralmente opuesto a REST.

REST ha ganado popularidad frente a SOAP, especialmente en integración de servicios entre organizaciones y microservicios. Una API diseñada bajo principios REST se denomina **RESTful**.

Las API RESTful favorecen enfoques sencillos con:

- Menor generación de código y herramientas automatizadas.
- Uso de formatos como OpenAPI (Swagger) para describir APIs y producir documentación.

CAPÍTULO 2: ESTADO DEL ARTE

Los artículos que se muestran a continuación, abordan técnicas de visión artificial y aprendizaje profundo para la solución de problemáticas similares a las planteadas en esta investigación:

Empezando con la investigación de Tamayo-Monsalve et al. [29], que probaron un sistema de adquisición de imágenes multispectrales con diferentes arquitecturas de redes neuronales convolucionales. Evaluaron cinco modelos diferentes sobre un conjunto de datos balanceado, alcanzando una exactitud del 98 % en la clasificación de cinco estados de madurez, como se muestra en la figura 24. Los autores concluyeron que este enfoque de visión por computadora ofrece a los agricultores un método preciso y confiable para optimizar la cosecha, apoyando así las prácticas de agricultura de precisión.

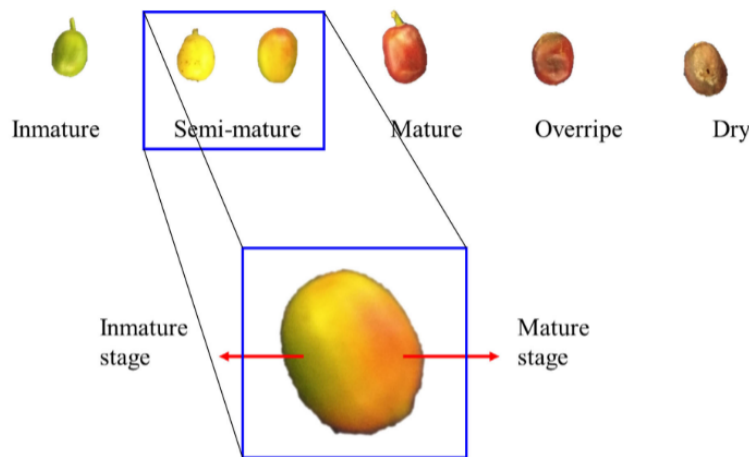


Figura 24: Estados de madurez de cerezas de café. Fuente: Tamayo-Monsalve et al. [29, Figura 1].

En la investigación de Bazame et al. [30] se centraron en la detección de cerezas de café en ramas en zonas de cultivo, cuyo conjunto de datos fue obtenido mediante fotografías de celular. Compararon las arquitecturas YOLOv3, YOLOv4 y sus versiones "tiny", como se muestra en la figura 25, para detectar y clasificar las cerezas en tres etapas de madurez. El modelo YOLOv4 demostró el mejor rendimiento con un 81 % de mAP para imágenes con resolución de 800x800 píxeles, siendo el modelo más robusto y el menos influenciado por imágenes con

alta densidad de objetos.

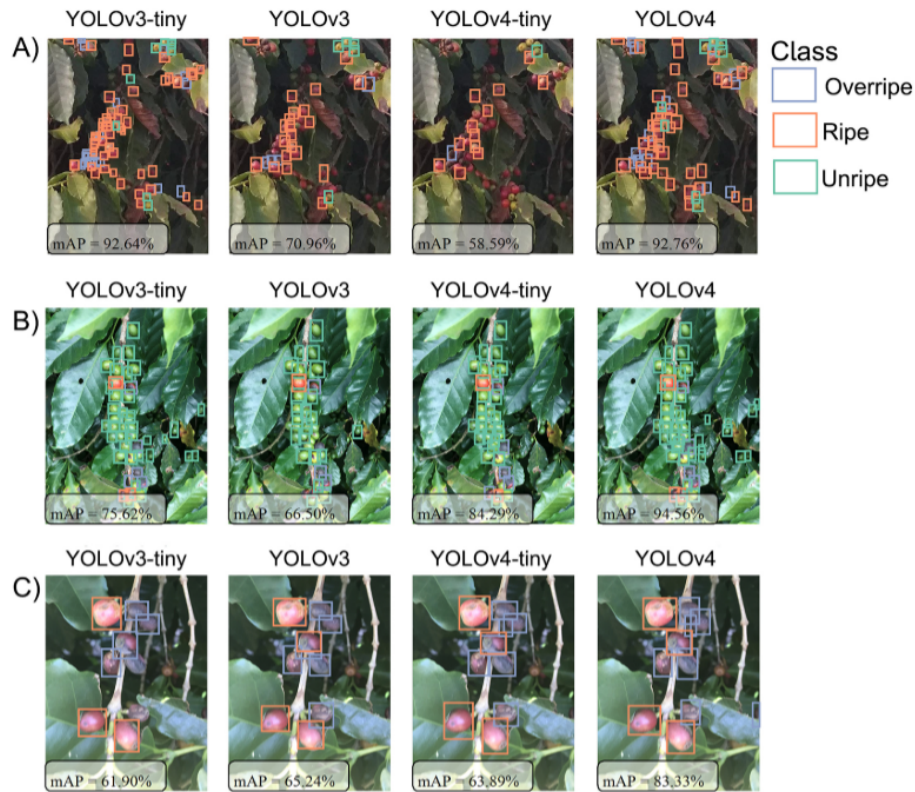


Figura 25: Detecciones de frutos de café realizadas por algoritmos YOLO considerando un tamaño de red de 800×800 píxeles para tres imágenes arbitrarias que representan frutos (A) en la sombra, (B) entre las hojas y (C) superpuestos. Fuente: Bazame et al. [30, Figura 3].

También, en Fuentes et al. [31] implementaron un sistema de visión artificial con redes neuronales que clasifica frutos de café en ‘maduros’ o ‘no maduros’, como se muestra en la figura 26, con 97.6% de precisión en clasificación aislada con luz ambiente controlado utilizando solo 196 imágenes, demostrando que este método es una solución para la reducción de costos laborales y tiempo, además de mejorar la calidad del producto final del café.



Figura 26: Pruebas en árbol. Fuente: Fuentes et al. [31, Figura 5].

Con Mimma et al. [5] abordaron la clasificación y detección de frutas combinando el uso de arquitecturas como YOLOv3 para la detección de objetos y ResNet50/VGG16 para la clasificación, siendo validadas en conjuntos de datos tanto públicos como privados. Logrando una precisión del 86 % y del 85 % con ResNet50 y VGG16 en el conjunto de datos público, respectivamente, mientras que en el conjunto de datos personalizado, lograron una precisión del 99 % con ResNet50 y del 98 % con el modelo VGG16. Después, desplegaron los modelos en un sitio web y crearon una aplicación para Android, como se muestra en la figura 27, con el fin de ser utilizados en la industria, supermercados y herramienta educativa para los niños.

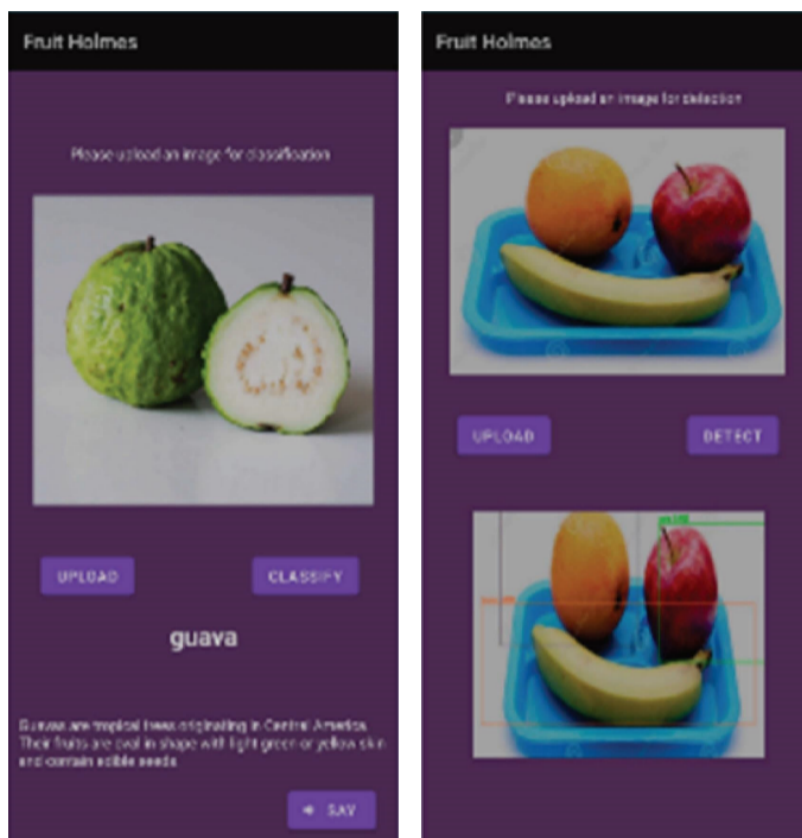


Figura 27: Ejemplo del trabajo de muestra de la interfaz de la aplicación para teléfonos inteligentes. Fuente: Mimma et al. [5, Figura 18].

Y con Madhavi et al. [32] se analizó las implementaciones recientes de aprendizaje profundo en la clasificación y detección de madurez en tomates, como se muestra en la figura 28, haciendo énfasis de técnicas como YOLOv3 para estimación de rendimiento en entornos reales. Destaca Faster R-CNN por su precisión en segmentación o detectores rápidos como SSD, y se propone métodos semisupervisados para reducir anotaciones. Concluyen que el uso de técnicas de aprendizaje profundo contribuyen al desarrollo agrícola sostenible al permitir la estimación automatizada del rendimiento, reducir los costos laborales y optimizar el momento de la cosecha.



Figura 28: Cajas delimitadoras y etiquetas de clases generado por la red. Fuente: Madhavi et al. [32, Figura 18].

Enfocados en el tema del café, fue la investigación de Anita y Albarda [33], que reportaron precisiones limitadas de 72.12 % con KNN frente a 24.41 % con ANN en la clasificación de cerezas de café, lo que demuestra la superioridad del algoritmo KNN para esta tarea en su estudio. De manera similar, en la investigación de Huang et al. [34] se implementó un sistema CNN para la clasificación en tiempo real de granos de café verdes, logrando una tasa de reconocimiento del 93 %.

En otras investigaciones se habla sobre el implemento de redes neuronales en edge computing, tal es el caso de Vilcamiza et al. [35], quienes desplegaron un clasificador de calidad de tueste en una NVIDIA Jetson Nano obteniendo 91.33 % de precisión. En la misma línea, Muhlisin et al. [6] desarrollaron una aplicación Android con ResNet50 para clasificar la madurez de granos de café tostados, reportando 100 % de precisión en pruebas controladas, aunque esta disminuyó a 83.3 % en escenarios reales. De la misma manera, Pei et al. [36] desarrollaron un sistema en tiempo real para clasificación de color en cerezas usando TensorFlow y Python.

Además de la madurez del fruto, esta tecnología se ha aplicado a otras problemáticas del cultivo, como en la investigación de Santana et al. [37], que se automatizó el conteo de plantas de café usando YOLOv3, con 96.8 % de exactitud a los seis meses. Por otro lado, Abreu Júnior et al. [38] utilizaron redes neuronales e imágenes multiespectrales de Sentinel-2 para estimar rendimientos de café con un alto coeficiente de determinación ($R^2 = 0.82$). También Lewis y Espineli [39] crearon un sistema CNN para detectar deficiencias nutricionales en hojas, mientras que Eron et al. [40] desarrollaron CoffeApp, una aplicación móvil con YOLOv7 para el monitoreo integral del cultivo. Incluso se han explorado sensores inteligentes de bajo consumo para procesamiento en el edge computing como en la investigación de Hsu et al. [41].

Si se deja al café a un lado y se hace más general el uso de redes neuronales en el cultivo, se encuentra investigaciones como la de Yang et al. [42], que alcanzó un 99.2 % de precisión en clasificar la madurez de plátano usando ResNet101, mientras que Xu et al. [43] logró un 99.7 % en clasificar semillas de maíz con P-ResNet, y Khan et al. [44] reconoció variedades de habichuela con un 97.5 % de precisión.

También Ko et al. [45] propusieron una metodología para la detección de madurez de tomate llamado SDF-ConvNets, en el cual combinaron múltiples redes CNN con fusión estocástica de decisiones. Procesando imágenes de 2,712 muestras, su método alcanzó un 96.5 % de F1-score en la clasificación de cinco etapas de madurez, demostrando su superioridad al ser comparado con métodos relacionados. De forma similar, Rivera y Reyes-Duke [46] aplicaron redes neuronales convolucionales con Roboflow para clasificar madurez de frutos de cacao, 92.3 % de mAP con imágenes capturadas a diferentes distancias, demostrando que el entrenamiento con variabilidad espacial mejora la robustez del modelo.

Más allá del cultivo de frutos, las redes neuronales se aplica a otros problemas de la agricultura, como se puede ver en estudios como los de Liu et al. [47] y Lottes et al. [48], que compararon métodos basados en color contra métodos con redes neuronales convolucionales para la detección de plantas y malezas, encontrando una mayor robustez en las arquitecturas de deep learning. Por otro lado, Ratnayake et al. [49] desarrollaron el algoritmo Polytrack para el monitoreo automatizado de polinización, alcanzando una precisión de 0.975 y un recall de

0.972 en el seguimiento de insectos. En el ámbito de la estimación de la floración, Darwin et al. [50] alcanzó un 92.51 % de precisión promedio.

La investigación de Jayaweera et al. [51] contribuyó con un dataset especializado llamado MangoDB para entrenar modelos en la clasificación de mango, mientras que Ahmed y Rafique [52] demostraron que la división de imágenes en parches pequeños mejora la segmentación pixel-wise de cultivos en escenarios complejos.

Además, Patricio y Rieder [53] analizaron 25 estudios sobre cultivos de granos, identificando el potencial de las CNN para la detección de plagas. En el caso de Fracarolli et al. [54], revisaron aplicaciones en toda la cadena agroalimentaria, destacando la integración con nuevos sensores. En el tema del meta-análisis, Pyingodi et al. [55] comentan con 122 tipos de frutas, el ResNet logró un 98 % de precisión, el aumento de imágenes no mejora significativamente el rendimiento, siendo la arquitectura del modelo el factor más crítico.

Finalmente, las redes neuronales para tareas de visión artificial es aplicado en otras áreas, tales se explican en la investigación de Ohura et al. [56] donde aplicaron U-Net para segmentación de heridas, la de Tsuboi et al. [57] donde detectaron angioectasias intestinales, y la de Ziyad et al. [58] la cual clasificaron Alzheimer con CNNs.

CAPÍTULO 3: RED NEURONAL CONVOLUCIONAL PARA LA DETECCIÓN DE MADUREZ DE CEREZA DE CAFÉ

En este capítulo se describirá la metodología del desarrollo del modelo de red neuronal convolucional (CNN), tal como lo recomienda Ultralytics [59], para la detección y clasificación de madurez de cerezas de café y se analizarán sus resultados. En la figura 29 se muestra un diagrama de bloques del desarrollo del sistema clasificador de madurez de cerezas de café.

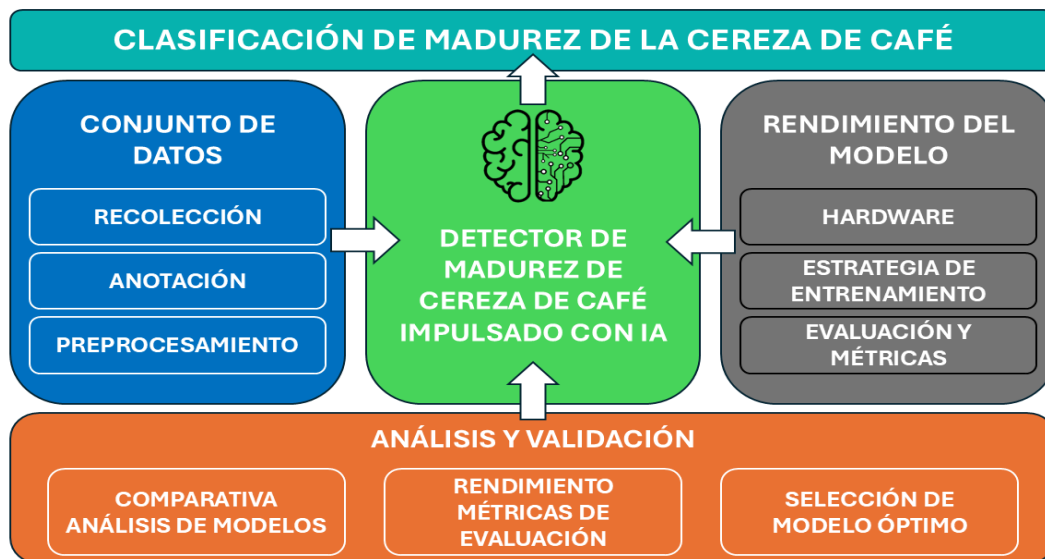


Figura 29: Metodología para la detección de madurez de cerezas de café

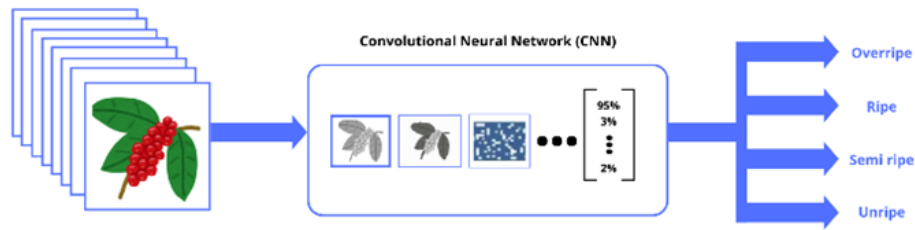
3.1 METODOLOGÍA

3.1.1 Selección de arquitectura

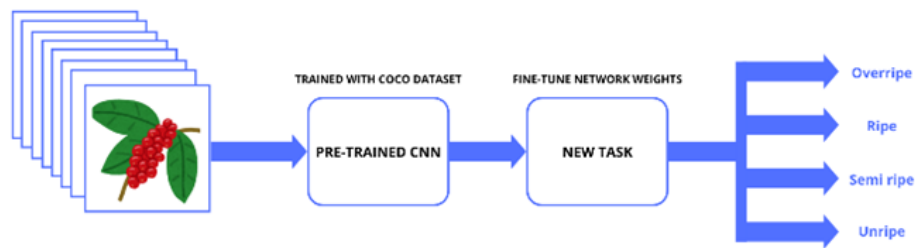
Las redes neuronales convolucionales (CNN) han sido usualmente utilizadas para tareas de visión computacional, como la clasificación y detección de objetos en imágenes, es por ello que se optó por utilizar un modelo CNN para la detección de madurez de cereza de café, en particular, un modelo con la arquitectura YOLO de la compañía Ultralytics. La razón por la que se usa la arquitectura YOLO es por los resultados que se comentan en las tesis de Manrique Santos [60] y Alvarado Escua [61], donde comparan varias arquitecturas y concluyen que la arquitectura YOLO obtiene mejores resultados en las métricas comparadas.

Al buscar el modelo YOLO adecuado para esta investigación, se eligieron 3 versiones de YOLO que pudieran ser utilizadas: YOLOv8 (la version más utilizada en artículos de investigación), YOLOv9 (una versión mejorada de YOLOv8) y YOLO11 (la versión más reciente que ha desarrollado Ultralytics y que supera en rendimiento a los anteriores de acuerdo a lo revisado en la sección de [YOLO11](#) en el capítulo anterior).

Dicho lo anterior, se seleccionó el modelo YOLO11 para el desarrollo de la investigación y se continuó eligiendo el tipo de entrenamiento. Este puede realizarse de dos formas, con el entrenamiento From Scratch, como se observa en la figura [30A](#), que requiere un conjunto de datos diverso para construir la comprensión del modelo desde la base, como lo menciona Ultralytics [\[59\]](#), o con el entrenamiento Transfer Learning, como se presenta en la figura [30B](#) que, en lugar de inicializar aleatoriamente los pesos y sesgos de la red neuronal, utiliza los valores obtenidos en un modelo pre-entrenado y con esto, adaptarlo a un conjunto de datos específico como lo menciona Géron [\[10\]](#) en su libro *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*.



(a)



(b)

Figura 30: Tipos de entrenamiento en un modelo YOLO: (a) Training from Scratch y (b) Transfer Learning.

En esta investigación se eligió utilizar el entrenamiento por Transfer Learning, siendo esta técnica la más ahorradora de tiempo y la que requiere de menos datos al entrenar como lo menciona Géron [10] en su libro mencionado previamente.

3.1.2 Recolección de datos.

Se conformó un amplio conjunto de imágenes de cerezas de café donde se contenía los cuatro estados de madurez objetivos de esta investigación: verde, semi-maduro, maduro y sobre-maduro. Para garantizar la funcionalidad del modelo al ser ejecutado en zonas de cultivo, en el conjunto de imágenes se recolectó imágenes que tuvieran variación en los fondos e iluminación como se encontraría en la realidad; en la Figura 31, se muestra un ejemplo por cada una de las diferentes tonalidades de la cereza de café que detectará el modelo YOLO

para determinar la etapa de maduración de la misma.

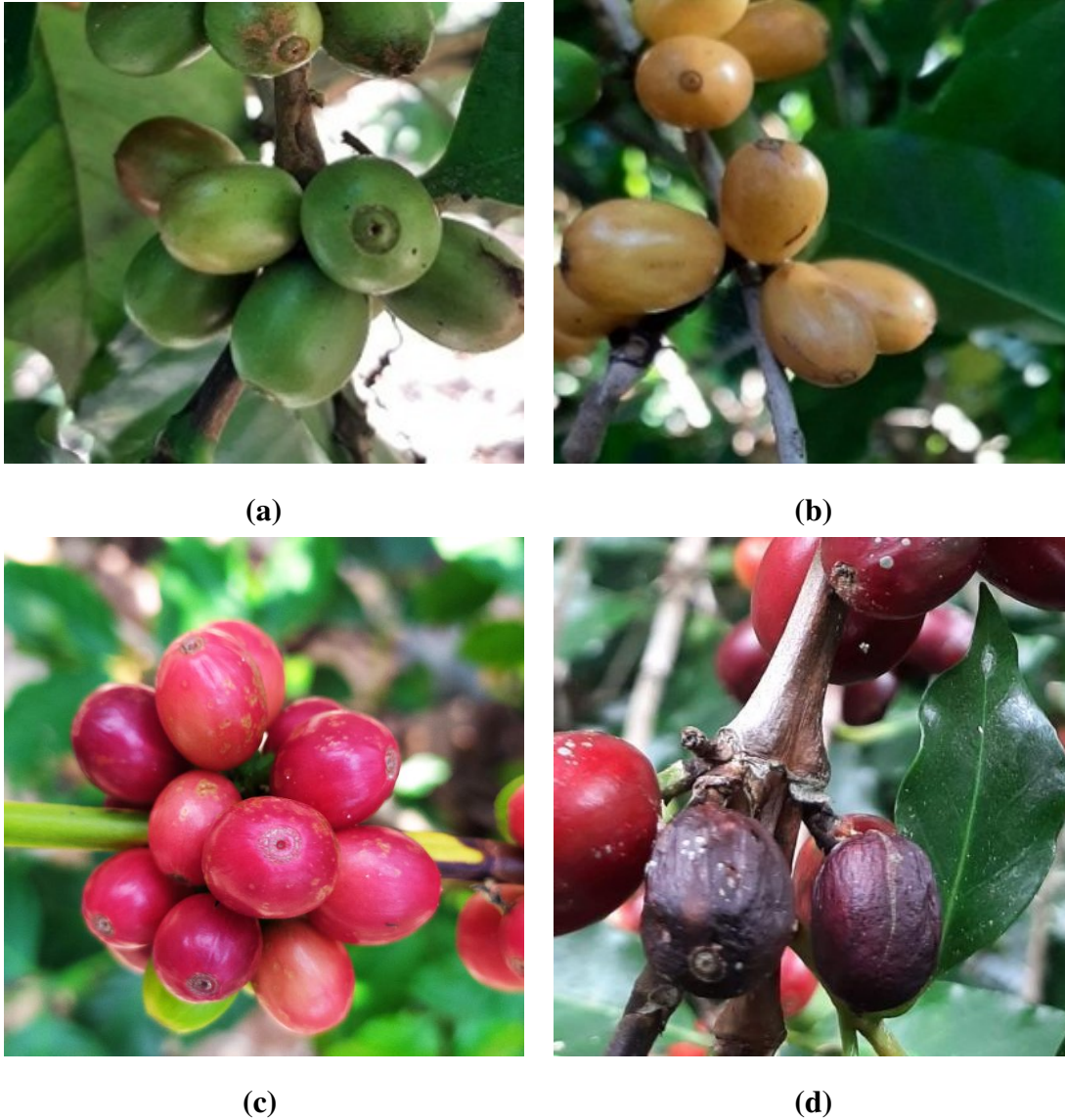


Figura 31: Estados de madurez de la cereza de café: (a) Verde, (b) Semi-maduro, (c) Maduro y (d) Sobre-maduro.

Considerando la importancia de la calidad y cantidad del conjunto de datos en el rendimiento del modelo de acuerdo a Géron [10], se optó por combinar varios conjuntos de datos públicos existentes para conformar una colección más robusta.

Se utilizó la interfaz web Roboflow [62] para unificar los conjuntos de datos de cerezas de café que se seleccionaron desde los repositorios públicos de Kaggle [63], Roboflow [64], Roboflow

[65] y Roboflow [66] en los conjuntos de datos llamados *Coffee Fruit Maturity*, *Coffee Cherry Beans Detection Computer Vision Model*, *Coffee Dataset 2 Computer Vision Dataset* y *coffee fruit Computer Vision Dataset* respectivamente.

Dentro de la interfaz web Roboflow, se ingresó a la sección **Projects** y se creó un nuevo proyecto de detección de objetos, escribiendo los grupos de anotación (verde, semi-maduro, maduro y sobre-maduro), como se muestra en la figura 32. Posteriormente, se importaron los tres conjuntos de datos mencionados anteriormente, los cuales contuvieron un total de 7,778 imágenes.

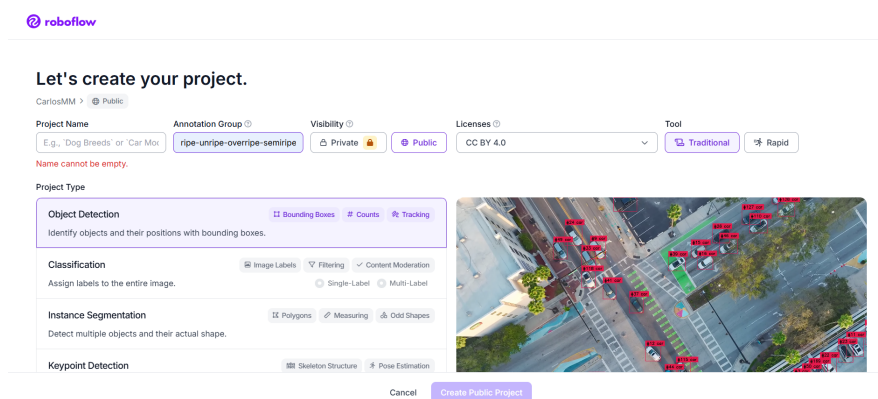
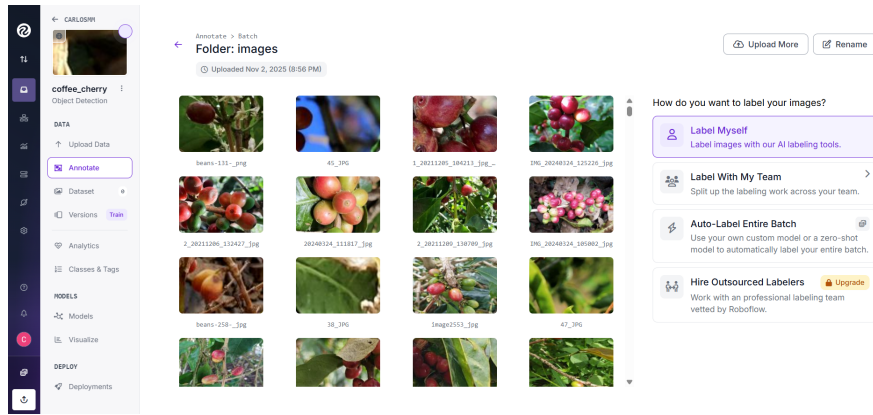


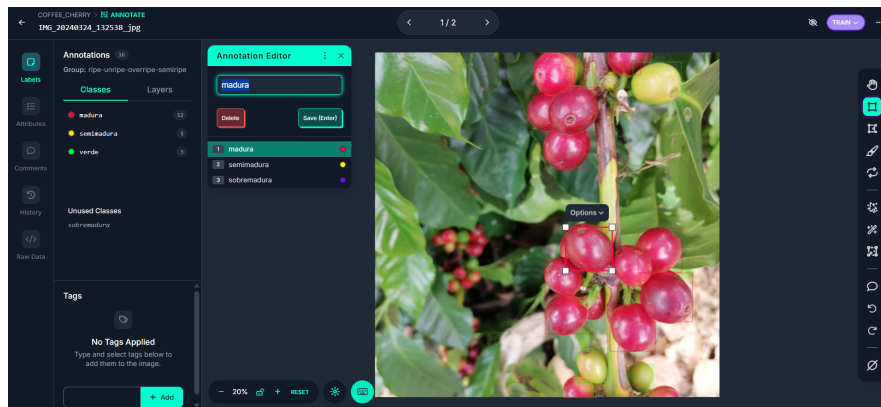
Figura 32: Captura de pantalla de la creación de un proyecto en Roboflow.

3.1.3 Anotación de datos.

En la sección **Annotate** de la interfaz web Roboflow, como se muestra en la figura 33A, se realizó el proceso de anotación de las imágenes donde todas las cerezas de café que aparecen en las imágenes fueron marcadas manualmente mediante cajas delimitadoras, tal como se muestra en la Figura 33B, para determinar la posición de cada cereza dentro de la imagen y asignar su clasificación correspondiente (verde, semi-maduro, maduro, sobre-maduro), siendo esto un aspecto importante para que el modelo pueda aprender la relación entre entradas y salidas, como lo comenta Ultralytics [67]. Una vez finalizado, se debe seleccionar la opción de **Split Images Between Train/Valid/Test** para dividir el conjunto de datos anotados en tres subconjuntos: entrenamiento, validación y prueba, tal como lo comenta la documentación de Roboflow [68].



(a)

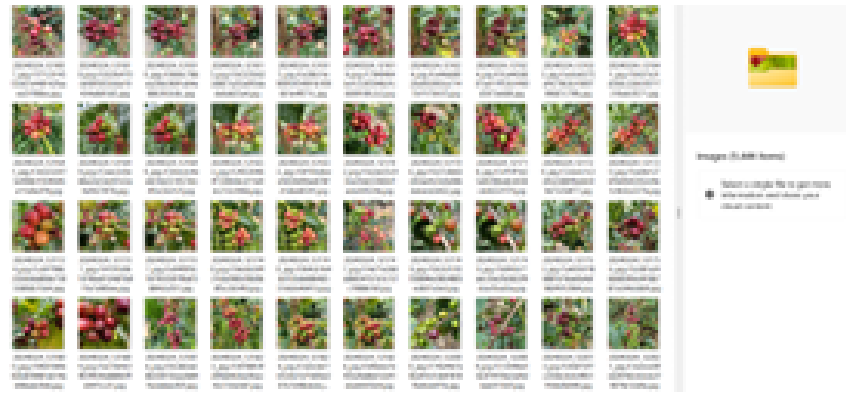


(b)

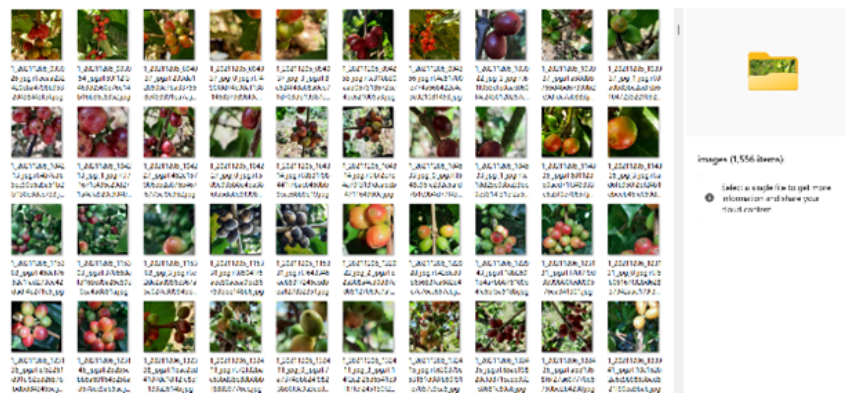
Figura 33: Proceso de etiquetado: (a) Captura de pantalla de la sección ‘Annotate’ en Roboflow y (b) Etiquetado de imágenes con cajas delimitadoras.

3.1.3.1 División de datos.

Como se mencionó anteriormente, el conjunto de datos con 7,778 imágenes se dividió en tres subconjuntos donde, siguiendo la documentación de Roboflow [69], se contuvo en el subconjunto de entrenamiento el 70 % del conjunto de datos con 5,446 imágenes en total (Figura 34A), el subconjunto de validación el 20 % con 1,556 imágenes (Figura 34B) y el subconjunto de prueba el 10 % con 776 imágenes (Figura 34C). El subconjunto de entrenamiento es donde el modelo aprenderá los parámetros, el de validación es el que se usará para guiar la selección de hiperparámetros y el de prueba es el que se usará para estimar el error generalizado después del entrenamiento, de acuerdo a Goodfellow et al. [70]. Para una mejor visualización, se puede acceder al conjunto de datos utilizado en esta investigación, en el siguiente [enlace](#).



(a)



(b)



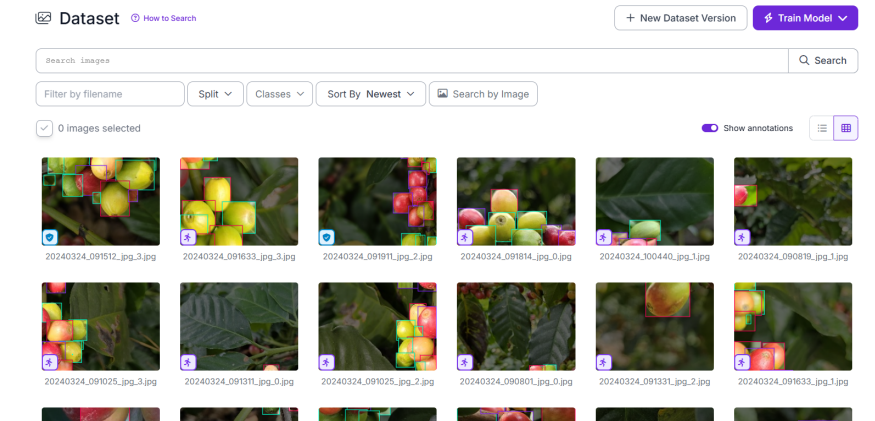
(c)

Figura 34: Capturas de pantalla del conjunto de datos: (a) Subconjunto de entrenamiento, (b) Subconjunto de validación y (c) Subconjunto de prueba.

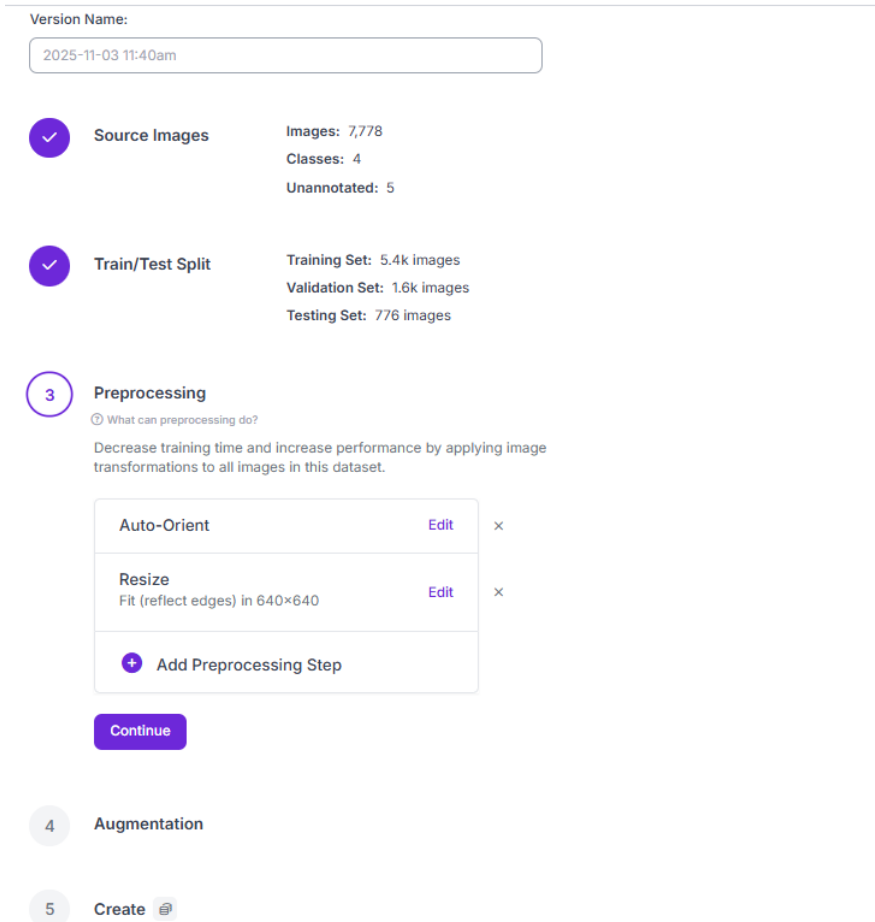
3.1.4 Preprocesamiento de datos.

Después de anotar las imágenes del conjunto de datos y dividirlos, en la sección **Dataset** de la interfaz web Roboflow se seleccionó el botón **New Dataset Version**, como se muestra en la figura 35A, para generar una version nueva del conjunto de datos y preparar las imágenes con las etiquetas para el entrenamiento del modelo.

A continuación, se mostró una nueva pantalla llamada **Versions** donde se puede ver la configuración para la nueva versión del conjunto de datos, como se observa en la figura 35B, donde se debe de asegurar que la separación de los subconjuntos de entrenamiento, validación y prueba mantenga los porcentajes 70 %, 20 % y 10 %, como se explicó en la subsubsección anterior.



(a)



(b)

Figura 35: Captura de pantalla de la versión del conjunto de datos en Roboflow: (a) Interfaz de Roboflow en la sección Dataset y (b) Interfaz de Roboflow en la sección Versions.

Una vez verificado, en el apartado del preprocesamiento se seleccionó la opción de **Auto-Orient**, como se muestra en la figura 36A, que se encarga de corregir automáticamente la rotación de la imagen para una visualización correcta. La segunda opción fue el **Resize**, haciendo uso de la opción **Fit (reflect edges) in 640x640**, como se muestra en la figura 36B, el cual ajusta todas las imágenes a las dimensiones de entrada requeridas por el entrenamiento del modelo YOLO, que en este caso son 640 píxeles por 640 píxeles.

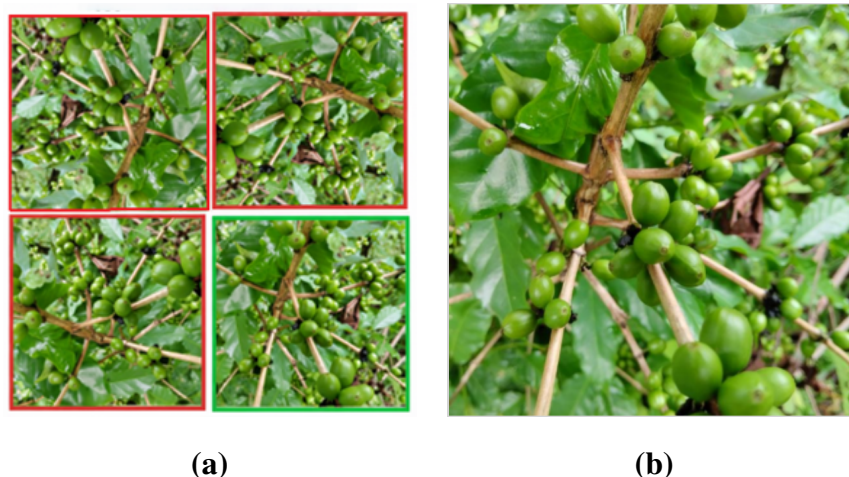


Figura 36: Preprocesamiento de imágenes: (a) Auto-Orient y (b) Resize.

3.1.4.1 Aumento de datos.

Con el objetivo de incrementar el número de imágenes para el conjunto de datos, se aplicó el proceso de aumento de datos en el subconjunto de entrenamiento para ayudar al modelo a identificar mejor las cerezas de café, tal como se recomienda realizar en Roboflow [71].

El siguiente apartado dentro de la pantalla **Versions** de Roboflow, se llama **Augmentation**, donde las opciones de aumento de datos que se usaron fueron Rotation (Figura 37A) con un valor entre -15° y $+15^{\circ}$, Brightness (Figura 37B) entre -20% y $+20\%$, y Blur (Figura 37C) de hasta 2.5 píxeles.

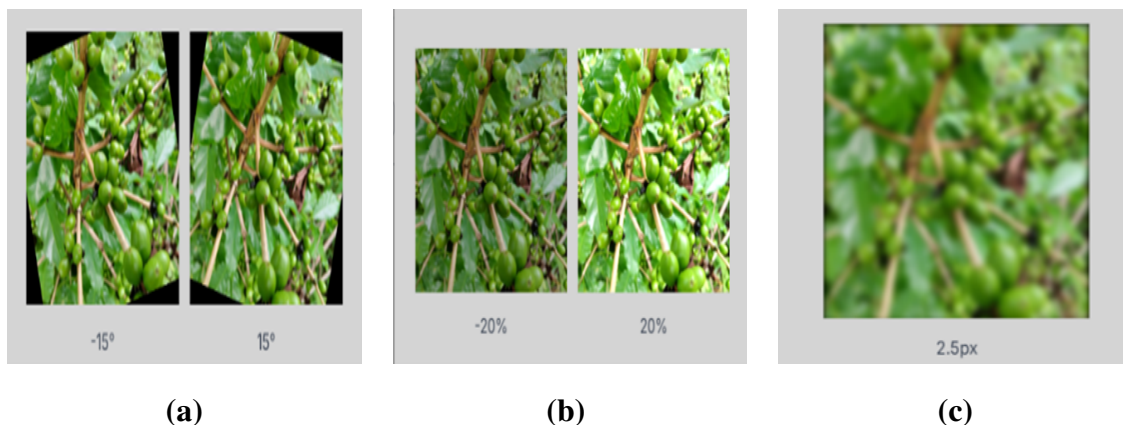


Figura 37: Aumento de datos: (a) Imagen con Rotation, (b) Imagen con Brightness y (c) Imagen con Blur.

Por último, en el apartado **Create** se creó el conjunto de datos que contiene las imágenes con los aumentos seleccionados donde, por cada imagen original del subconjunto de entrenamiento se obtuvieron tres nuevas variantes gracias a los aumentos, como lo menciona Roboflow [71], lo que ocasionó la expansión de este subconjunto, pasando de 5,446 a 16,337 imágenes. El nuevo conjunto de datos contuvo un total de 18,669 imágenes, donde el subconjunto de entrenamiento tenía el 88 % con 16,337 imágenes, el subconjunto de validación el 8 % con 1,556 imágenes y el subconjunto de prueba el 4 % con 776 imágenes. En la figura 38 se muestra una captura de pantalla de la interfaz web de Roboflow donde se aprecia la distribución de cada subconjunto, las técnicas de preprocesamiento y aumento que se le aplicó al conjunto de datos.

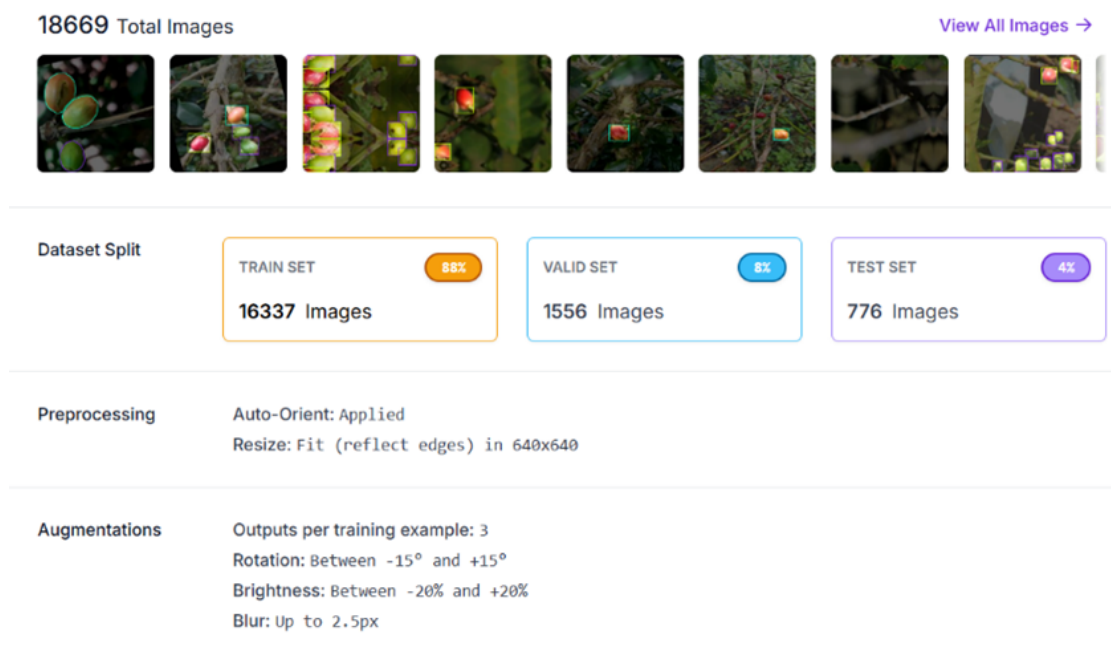


Figura 38: Captura de pantalla de la distribución del conjunto de datos en Roboflow.

Con esta nueva version del conjunto de datos realizada, se descargó en formato YOLO11, para que pudiera ser compatible con el entrenamiento del modelo.

3.1.5 Entrenamiento y Evaluación.

El entrenamiento del modelo y las siguientes etapas de la investigación para la detección de madurez de la cereza del café (validación, predicción y ajuste fino), que se explicarán en la sección 3.2 de este capítulo, fue implementado con el lenguaje de programación Python, ya que proporciona facilidad en su uso y permite trabajar con el framework de Ultralytics. Al hacer uso de Anaconda como gestor de paquetes y entornos, se utilizó Jupyter Notebook después de crear un entorno virtual, para escribir y ejecutar los códigos de Python.

Esta etapa se realizó en una computadora local en lugar de trabajar en la nube para aprovechar la capacidad de cómputo disponible en el laboratorio de automatización del Tecnológico Nacional de México Campus Orizaba y evitar costos operativos. Dicha estación de trabajo estaba equipada con una GPU NVIDIA GeForce RTX 3080 Ti (12 GB VRAM), una CPU de 24 núcleos y 128 GB de RAM. El entorno de software se configuró con Python 3.10.18,

PyTorch 2.5.1+cu121 y la biblioteca Ultralytics 8.3.152.

Durante el proceso de entrenamiento, el modelo aprendió con el conjunto de datos con aumentos durante un número determinado de épocas, utilizando el método de retropropagación, el cual ajusta los parámetros internos al comenzar con el valor de pérdida final y después trabajando hacia atrás desde las capas superiores a las inferiores, aplicando la regla de la cadena para calcular la contribución de cada parámetro en el valor de pérdida, tal como menciona Chollet [72] en su libro *Deep Learning with Python*.

De acuerdo con la documentación oficial de Ultralytics [13], existen cinco tamaños diferentes del modelo YOLO11, los cuales son: YOLO11n (nano), YOLO11s (pequeño), YOLO11m (mediano), YOLO11l (grande) y YOLO11x (extra grande). Cada uno de ellos tienen diferentes capacidades de cómputo y de precisión, siendo YOLO11n el modelo más ligero pero más impreciso y YOLO11x el más pesado pero más preciso. Por lo tanto, el proceso de entrenamiento y evaluación se le aplicó únicamente a tres tamaños del modelo YOLO11: YOLO11n, YOLO11m y YOLO11x, con el fin de analizar el rendimiento de cada uno y seleccionar el más adecuado para la presente investigación.

Por lo que, dentro de Jupyter Notebook se ejecutó el código de Python 3.1 para el entrenamiento de los tres tamaños del modelo YOLO11, como se muestra a continuación.

Listing 3.1: Entrenamiento del modelo YOLO11

```
1      # Importar la libreria Ultralytics
2      from ultralytics import YOLO
3
4      # Cargar el modelo
5      model = YOLO("yolo11m.pt")
6
7      # Entrenamiento del modelo
8      results = model.train(
9          data = r"C:\Users\labau\Documents\carlos_miranda\yolo\
          Dataset\dataset_v15_yolov11\data.yaml",
10         epochs = 60,
```

```

11     batch = 12
12     imgsz = 640,
13     device = "0",
14     optimizer = 'AdamW',
15     name = "entrenamiento_yolov11m",
16     amp=False,
17     lr0 = 0.001,
18     weight_decay = 0.0005,
19     patience = 10
20 )

```

Estos hiperparámetros se configuraron con el fin prevenir el overfitting y obtener el mejor resultado posible en el entrenamiento, siguiendo la documentación oficial de Ultralytics [73]. A continuación, se explicará los hiperparámetros seleccionados:

- `epochs = 60` donde cada época representa un pase completo sobre todo el conjunto de datos.
- `batch = 12` que es el tamaño de lote al cual las muestras de entrenamiento son procesadas antes de que se actualicen los parámetros internos del modelo.
- `imgsz = 640` que es el tamaño de imagen objetivo para el entrenamiento, en este las imágenes se redimensionan a cuadrados con lados iguales al valor especificado.
- `device = 0` que especifica a la GPU como el dispositivo computacional para el entrenamiento.
- `optimizer = 'AdamW'` el cual establece a AdamW como el optimizador del entrenamiento, afectando directamente a la velocidad y la estabilidad de la convergencia.
- `amp = False` que desactiva el entrenamiento automático de precisión mixta para una mayor estabilidad del entrenamiento.
- `lr0 = 0.001` que es la tasa de aprendizaje, influyendo en la rapidez con la que se actualizan los pesos del modelo.

- `weight_decay = 0.0005` que es la regularización mediante decaimiento de pesos, el cual penaliza los pesos grandes para evitar el overfitting.
- `patience = 10` el cual espera 10 épocas para detener el entrenamiento cuando el rendimiento se estanca.

Al haber ejecutado el código 3.1, se entrenó a los tres tamaños del modelo YOLO11 que se tomaron en cuenta y automáticamente se le evaluó en cada época. En la etapa de la evaluación se utilizó el subconjunto de validación, que contiene 1,556 imágenes, con el fin de analizar el comportamiento del modelo al detectar cerezas de café, gracias a las métricas numéricas y salidas visuales que se proporcionan al finalizar la validación. Por otro lado, se utilizan las métricas de validación para plantear un posible ajuste en el modelo y obtener un mejor rendimiento, tal como lo menciona la documentación de Ultralytics [74].

3.2 RESULTADOS Y DISCUSIONES.

3.2.1 Análisis de resultados.

El rendimiento de los modelos se evaluó principalmente mediante la métrica de precisión media promedio (mAP), el cual se divide a su vez en dos métricas, el mAP@0.5 que usa el valor predeterminado de 0.5 para la intersección sobre unión, como lo menciona Lin et al. [75] en su artículo «Microsoft COCO: Common objects in context,» y mAP0.5:0.95 siendo una métrica más detallada y estricta donde los umbrales de intersección sobre unión son desde 0.5 hasta 0.95, como lo menciona la documentación de Ultralytics en Ultralytics [76]. Adicionalmente, se analizó el valor de Precisión, la cual mide la exactitud de las predicciones positivas del modelo y de Recall, que mide la proporción de casos positivos reales que el modelo identifica correctamente como lo menciona Olson y Delen [77] en su artículo *Advanced Data Mining Techniques*.

El modelo YOLO11n, como se observa en la tabla 1, alcanzó valores generales en la precisión de 0.775 y recall de 0.785, con un mAP50 de 0.841 y un mAP50-95 de 0.706. Observando los valores de los resultados por clase, la clase madura presentó el valor más alto de Recall con 0.825, lo que significa que el modelo detecta más objetos de esta clase y el valor más bajo del

mismo fue sobre-madura con 0.722. Por otro lado, el valor de precisión más baja de las cuatro clases es la sobre-madura con 0.723, lo que puede resultar en más falsos positivos y el valor más alto con 0.816 fue inmadura. Además, se puede observar que el valor más alto de mAP50 pertenece a la clase inmadura con 0.874 y la más baja con 0.796 es de la clase sobre-madura.

Tabla 1: Métricas de validación del modelo YOLO11n

Clase	Imágenes	Instancias	Precisión (P)	Recall (R)	mAP@50	mAP@50-95
General	1,556	16,880	0.775	0.785	0.841	0.706
Sobre-madura	466	1,893	0.723	0.722	0.796	0.705
Madura	878	6,095	0.772	0.825	0.847	0.715
Semi-madura	1,096	3,685	0.789	0.775	0.847	0.734
Inmadura	879	5,207	0.816	0.818	0.874	0.670

El modelo YOLO11m, como se muestra en la tabla 2, mostró un desempeño general con una precisión de 0.773, un recall de 0.806, y valores de mAP@50 y mAP@50-95 de 0.849 y 0.726 respectivamente. Al analizar los resultados por clase, se observó que la clase inmadura fue la que obtuvo los mejores valores en precisión con 0.840, recall con 0.823 y mAP@50 con 0.881. La clase que tuvo peores resultados fue la sobre-madura con 0.758 de precisión, 0.698 de recall y 0.794 mAP@50.

Tabla 2: Métricas de validación del modelo YOLO11m

Clase	Imágenes	Instancias	Precisión (P)	Recall (R)	mAP@50	mAP@50-95
General	1,556	16,880	0.773	0.806	0.849	0.726
Sobre-madura	466	1,893	0.698	0.758	0.794	0.720
Madura	878	6,095	0.783	0.830	0.856	0.732
Semi-madura	1,096	3,685	0.788	0.795	0.864	0.762
Inmadura	879	5,207	0.823	0.840	0.881	0.690

El modelo YOLO11x, como se muestra en la Tabla 3, obtuvo un desempeño general con

valores de precisión de 0.776, recall de 0.804, y mAP@50 de 0.847 y mAP@50-95 de 0.727. La clase donde alcanzó mejor valor en precisión, recall y mAP@50 fue en inmaduro con 0.841, 0.831 y 0.884 respectivamente. Por otro lado, la clase sobre-madura fue la que presentó resultados más bajos, con valores de 0.751 de precisión, 0.706 de recall y 0.789 de mAP@50.

Tabla 3: Métricas de validación del modelo YOLO11x

Clase	Imágenes	Instancias	Precisión (P)	Recall (R)	mAP@50	mAP@50-95
General	1,556	16,880	0.776	0.804	0.847	0.727
Sobre-madura	466	1,893	0.706	0.751	0.789	0.720
Madura	878	6,095	0.781	0.833	0.853	0.731
Semi-madura	1,096	3,685	0.786	0.792	0.862	0.763
Inmadura	879	5,207	0.831	0.841	0.884	0.693

La comparación de los tres modelos YOLO11n, YOLO11m y YOLO11x, como se muestra en la tabla 4, permite observar las pérdidas durante el entrenamiento, se observa que YOLO11m y YOLO11x alcanzaron valores más bajos de box_loss con 0.67026 y 0.66516, respectivamente, en comparación con YOLO11n, obteniendo un valor de 0.72056, lo cual indica que el modelo tiene una peor capacidad de ajuste en la localización de las cajas delimitadoras a comparación de los demás modelos. Con respecto a cls_loss, que refleja la capacidad del modelo para asignar correctamente la clasificación al objeto detectado, el modelo YOLO11m presentó el valor más bajo con 0.5292, el modelo YOLO11x un valor de 0.53054 y el modelo que tuvo mayor valor en esta métrica con 0.58838 fue YOLO11n, lo que significa que este último modelo se equivoca más al identificar la categoría del objeto detectado.

Tabla 4: Comparativa de métricas de resultados entre modelos YOLO11

Métrica	YOLO11n	YOLO11m	YOLO11x
Pérdidas de entrenamiento			
train/box_loss	0.72056	0.67026	0.66516
train/cls_loss	0.58838	0.52920	0.53054
train/dfl_loss	0.98986	0.97947	1.01803
Métricas de evaluación			
metrics/precision(B)	0.77585	0.77573	0.77623
metrics/recall(B)	0.78418	0.80572	0.80368
metrics/mAP50(B)	0.84097	0.84690	0.84709
metrics/mAP50-95(B)	0.70629	0.72778	0.72674
Pérdidas de validación			
val/box_loss	0.66601	0.63490	0.62842
val/cls_loss	0.62662	0.58151	0.58621
val/dfl_loss	0.95718	0.96189	1.00876

Por otro lado, dfl_loss, es decir la predicción más precisa de los bordes de las cajas delimitadoras, los valores en los tres modelos se comportaron muy similar, aunque YOLO11n con un valor de 0.98986 y YOLO11m con valor de 0.97947, registraron un desempeño mejor que YOLO11x con un valor de 1.01803, ya que este último modelo tiende a predecir los bordes con menor precisión. En las métricas de precisión, recall y mAP, las diferencias entre los modelos fueron menos notorias. YOLO11x presentó el mayor valor de precisión con 0.77623, aunque la diferencia con YOLO11n con 0.77585 y YOLO11m con 0.77573 fue minima. En cuanto a la métrica de recall, YOLO11m obtuvo un valor de 0.80572, seguido por el modelo YOLO11x con 0.80368, mientras que YOLO11n fue el último con un valor de 0.78418. En cuanto a las métricas de mAP mostraron que YOLO11m y YOLO11x tuvieron un rendimiento superior en comparación a YOLO11n. El valor de mAP@50 fue de 0.8469 y 0.84709 para YOLO11m y YOLO11x, respectivamente, frente a 0.84097 para YOLO11n. Referente al valor

de mAP@50-95, YOLO11m alcanzó el mejor valor con 0.72778, ligeramente por encima de YOLO11x con 0.72674 y claramente superior al valor 0.70629 de YOLO11n.

En la etapa de validación, YOLO11n fue el modelo que obtuvo los valores más altos en box_loss y cls_loss con 0.66601 y 0.62662, respectivamente. En dfl_loss, YOLO11x alcanzó el valor más alto con 1.00876, lo que muestra la similitud del comportamiento de los modelos tanto en la etapa de entrenamiento como en la de validación. De acuerdo con las métricas observadas, aunque YOLO11x es menos preciso en el detalle de los bordes, logra identificar las cerezas de café con alta precisión y recall. Por su parte, YOLO11n tiene pérdidas más bajas en algunos aspectos, pero sus métricas finales son más bajas a comparación de YOLO11m y YOLO11x, lo que significa un menor rendimiento en general. Por lo que se optó por el modelo YOLO11m para realizar la investigación, ya que ofrecía una mejor precisión de detección que el modelo YOLO11n y un rendimiento comparable con menores requerimientos computacionales que el modelo YOLO11x.

Enfocándose en el modelo YOLO11m, además de la información numérica obtenida al finalizar la etapa de entrenamiento y evaluación, la función de validación genera salidas visuales para una mejor comprensión del rendimiento del modelo. Entre ellas se encuentran la matriz de confusión normalizada (Figura 39A), la versión normalizada que muestra los recuentos de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos para cada clase; la curva F1-Confidence (Figura 39B), ofrece información sobre el equilibrio del modelo entre falsos positivos y falsos negativos en diferentes umbrales; la curva Precision-Confidence (Figura 39C), ayuda a comprender cómo varía la precisión a medida que cambia el umbral; y la curva Precision-Recall (Figura 39D), muestra las ventajas y desventajas entre precisión y exhaustividad a diferentes umbrales; la curva Recall-Confidence (Figura 39E) ilustra cómo cambian los valores de exhaustividad en los diferentes umbrales; y por último, la gráfica de resultados (Figura 39F) permiten comparar la evolución del rendimiento del modelo durante el entrenamiento y validación a través de las épocas como lo menciona Ultralytics [78].

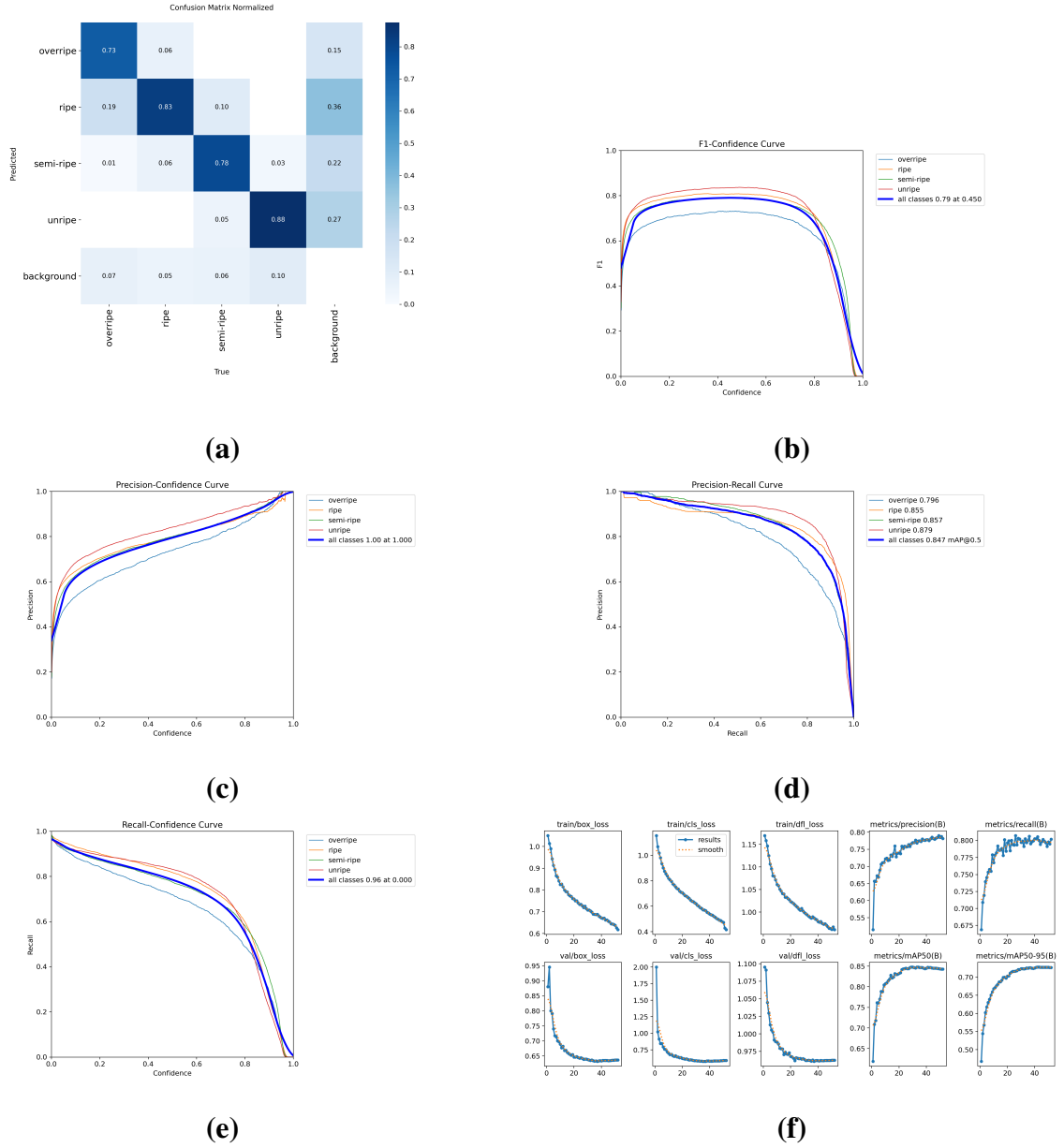


Figura 39: Salidas visuales del modelo YOLO11m: (a) Matriz de confusión normalizada, (b) Curva F1-Confidence, (c) Curva Precision-Confidence, (d) Curva Precision-Recall, (e) Curva Recall-Confidence y (f) Resultados.

3.2.2 Ajuste fino.

Tras analizar la etapa de la evaluación, se llevó a cabo el proceso de ajuste fino sobre el modelo YOLO11m mejor entrenado, con el propósito de mejorar su rendimiento en la detección de cerezas de café, como lo menciona la documentación oficial de Ultralytics Ultralytics [76], al

ejecutar el siguiente código de python 3.2.

Esta y las siguientes etapas se utilizó Google Colab para ejecutar los códigos en la nube, ya que la computadora local no se encontraba disponible. La GPU seleccionada fue una NVIDIA A100-SXM4-40GB, como se muestra en la figura 40.



Figura 40: Captura de pantalla de GPU seleccionada en Google Colab.

Listing 3.2: Ajuste fino del modelo YOLO11

```
1  # Importar la libreria Ultralytics
2  !uv pip install ultralytics
3  import ultralytics
4  ultralytics.checks()
5
6  #Vincular Google Drive
7  from google.colab import drive
8  drive.mount('/content/drive')
9
```

```

10 # Cargar el modelo
11 !yolo train
12 model = "/content/drive/MyDrive/Colab Notebooks/
    prueba_15_yolov11m_v15(3)--/weights/best.pt"
13
14 # Ajuste fino del modelo
15 data="/content/drive/MyDrive/Colab Notebooks/
    dataset_v15_yolov11/data.yaml"
16 epochs=50
17 batch=12
18 imgsz=640
19 seed=0
20 device="0"
21 optimizer='AdamW'
22 deterministic=True
23 amp=True
24 warmup_epochs=2
25 freeze=10
26 multi_scale=True
27 lr0=0.001
28 weight_decay=0.0005
29 patience=5
30 cos_lr=True
31 mosaic=0.5
32 mixup=0.1
33 name="ft_stage1(1)"

```

La configuración de los hiperparámetros para el ajuste fino, fueron los siguientes:

- epochs = 50.
- batch = 12.

- `imgsz = 640` que es el mismo tamaño de imagen utilizado en la fase del entrenamiento.
- `seed = 0` que establece la semilla aleatoria para el entrenamiento, asegurando la reproducibilidad de los resultados en ejecuciones con las mismas configuraciones.
- `device = 0` que especifica a la GPU como el dispositivo computacional para el entrenamiento.
- `optimizer = 'AdamW'` que establece a AdamW como el optimizador del entrenamiento.
- `deterministic = True` que fuerza el uso de algoritmos deterministas, asegurando la reproducibilidad.
- `amp = True` que activa el entrenamiento automático de precisión mixta.
- `warmup_epochs = 2` que es número de épocas para el calentamiento de la tasa de aprendizaje.
- `freeze = 10` que congela las primeras 10 capas del modelo, reduciendo el número de parámetros que se pueden entrenar.
- `multi_scale = True` que permite el entrenamiento a múltiples escalas aumentando/-disminuyendo `imgsz` hasta un factor de 0.5 durante el entrenamiento.
- `lr0 = 0.001` que es la tasa de aprendizaje, influyendo en la rapidez con la que se actualizan los pesos del modelo.
- `weight_decay = 0.0005` que es la regularización mediante decaimiento de pesos, el cual penaliza los pesos grandes para evitar el overfitting.
- `patience = 5` que espera 5 épocas para detener el entrenamiento cuando el rendimiento se estanca.
- `cos_lr = True` que habilita el programador de tasa de aprendizaje de coseno.
- `mosaic = 0.5` que es la probabilidad de combina 4 imágenes.

- `mixup = 0.1` que es la probabilidad de mezcla dos imágenes.
- `name = "ft_stage1(1)"` que es el nombre del experimento de ajuste fino.

Como se observa en la Figura 41, el ajuste fino provocó una mejora en los valores de la curva F1-Confidence (Figura 41A) en comparación con el modelo sin ajuste fino (Figura 39B). Sin embargo, las demás salidas visuales, las curvas Precision-Confidence (Figura 41B), Precision-Recall (Figura 41C), Recall-Confidence (Figura 41D), la matriz de confusión normalizada (Figura 41E) y los resultados de entrenamiento y validación (Figura 41F), mostraron un desempeño inferior respecto al modelo base. Este análisis indica que, tras el ajuste fino, el modelo no logró mejorar sus métricas globales como el mAP50 con un valor de 0.844 (Figura 39C) a diferencia del modelo base con un valor de mAP50 de 0.847 (Figura 39D), por el contrario, fueron valores menores a los esperados e incluso presentó indicios de overfitting. En consecuencia, se optó por continuar la investigación con el modelo YOLO11m sin ajuste fino.

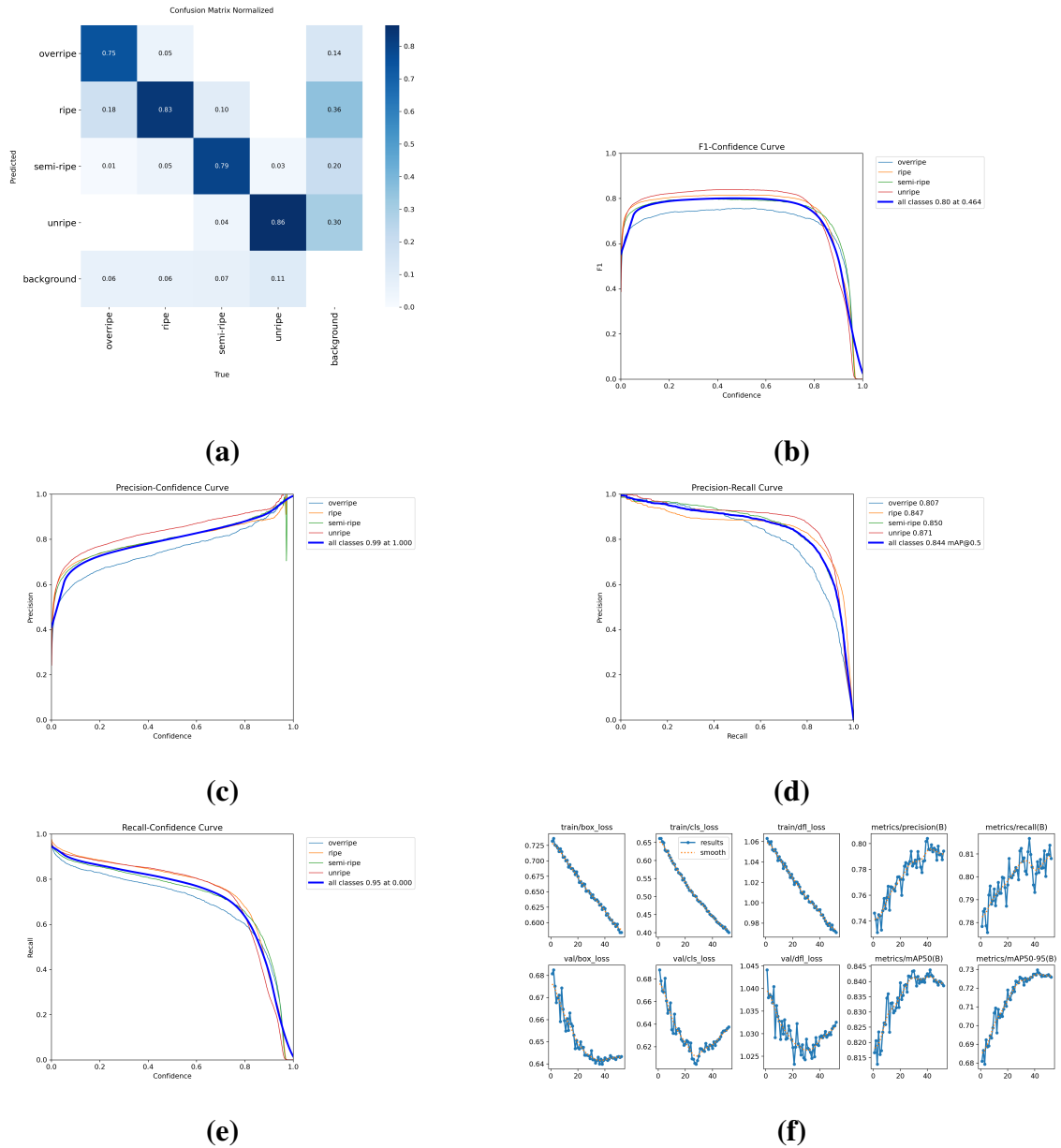


Figura 41: Resultados gráficos del ajuste fino en el modelo YOLO11m: (a) Matriz de confusión normalizada, (b) curva F1-Confidence, (c) curva Precision-Confidence, (d) curva Precision-Recall, (e) curva Recall-Confidence y (f) Resultados.

3.2.3 Prueba del modelo.

En esta etapa se utilizó el subconjunto de prueba, compuesto por 776 imágenes que no fueron utilizadas durante las fases anteriores, con el fin de comprobar que el modelo se comporta como se espera fuera del entorno de evaluación controlado, como lo menciona la documentación

oficial de Ultralytics Ultralytics [79]. Para probar el modelo YOLO11m entrenado se utilizó el modo de validación, con el fin de obtener una mayor evidencia del funcionamiento del modelo ante datos nuevos, por lo que, además de obtener las salidas visuales, se configuró la opción `save_json=True` dentro de los parámetros para almacenar todas las métricas de rendimiento en un archivo JSON, facilitando su posterior análisis, como se muestra en el código de python 3.3.

Listing 3.3: Prueba del modelo YOLO11

```
1  # Importar la libreria Ultralytics
2  from ultralytics import YOLO
3
4  #Vincular Google Drive
5  from google.colab import drive
6  drive.mount('/content/drive')
7
8  # Cargar el modelo
9  model = YOLO('/content/drive/MyDrive/Colab Notebooks/
10              prueba_15_yolov11m_v15(3)--/weights/best.pt')
11
12 # Prediccion del modelo
13 results = model.val(
14     data=r"C:\Users\labau\Documents\carlos_miranda\yolo\
15         Dataset\dataset_v15_yolov11\data.yaml",
16     # Para manejar tamanos de imagen variables
17     save_json=True, # Guarda metricas en JSON
18     name="val_prueba_15_yolov11m_v15(6)",
19     split="test"
20 )
```

El rendimiento del modelo durante su prueba se analizó mediante las métricas obtenidas en las salidas visuales. La matriz de confusión normalizada (Figura 42A) se observó con valores superiores los vistos en la validación del modelo, la curva F1-Confidence obtuvo un valor de 0.81 (Figura 42B), mientras que la curva Precision-Confidence presentó un valor de 1.00

(Figura 42C). En la curva Precision-Recall alcanzó un valor en mAP50 de 0.864 (Figura 42D) y la curva Recall-Confidence mostró un valor de 0.97 (Figura 42E). Por lo que se concluyó que el modelo mejoró su rendimiento al recibir nuevos datos de entrada.

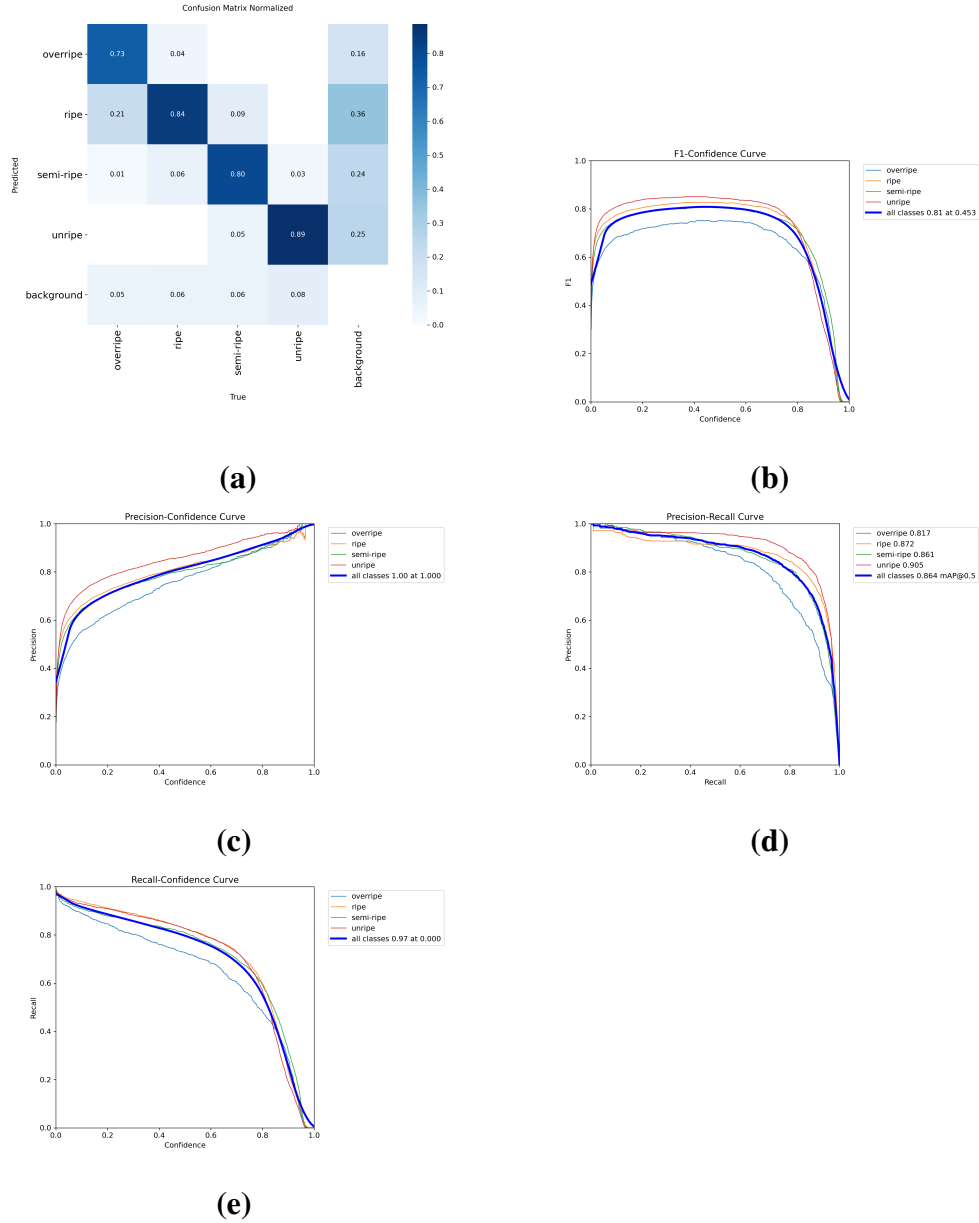


Figura 42: Resultados gráficos de la prueba del modelo YOLO11m: (a) Matriz de confusión normalizada, (b) curva F1-Confidence, (c) curva Precision-Confidence, (d) curva Precision-Recall y (e) curva Recall-Confidence.

Como se muestra en la Tabla 5, el modelo demostró realizar una buena ejecución en la

detección de la madurez de cereza de café, alcanzando un valor de mAP50 general del 86.4 %. La clase inmadura obtuvo el mayor valor de mAP50 (90.5 %), mientras que la clase sobre-madura presentó el menor valor (81.7 %). La métrica más estricta, mAP50-95, se situó en un valor general de 72.7 %, indicando una mejora en la precisión de la localización exacta de las cerezas de café.

Tabla 5: Resultados del modelo YOLO11m en el conjunto de validación

Clase	Imágenes	Instancias	Precisión (P)	Recall (R)	mAP@50	mAP@50-95
General	776	8,117	0.806	0.813	0.864	0.727
Sobre-madura	229	845	0.760	0.744	0.817	0.731
Madura	424	2,896	0.811	0.844	0.872	0.727
Semi-madura	538	1,830	0.798	0.822	0.861	0.747
Inmadura	441	2,546	0.856	0.843	0.905	0.703

Para finalizar este capítulo, se complementó la prueba utilizando el modo de predicción Ultralytics [80] con la librería de OpenCV en el código de Python 3.4 que se muestra a continuación, para ejecutar el modelo en imágenes donde predomine la categoría verde (Figura 43A), semi-madura (Figura 43B), madura (Figura 43C) y sobre-madura (Figura 43D), además de un video (Figura 43E) con el que se observaron las detecciones. Este modo no proporcionó métricas de rendimiento detalladas, pero permitió verificar de manera visual la precisión del modelo para la detección de cerezas de café con las cajas delimitadoras y las etiquetas de clasificación en imágenes del mundo real.

Listing 3.4: Predicción del modelo YOLO11

```

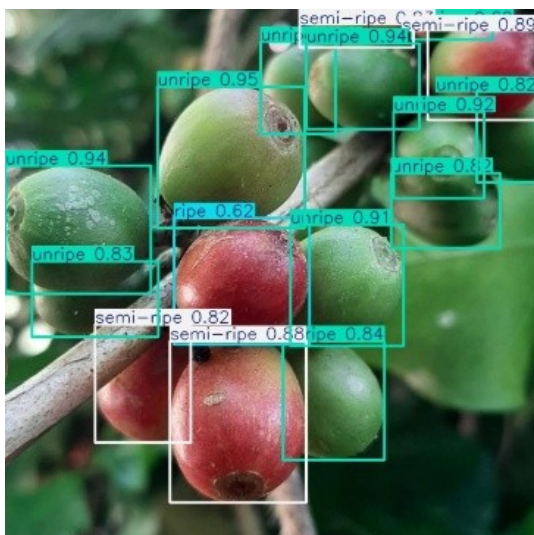
1  # Importar la libreria Ultralytics
2  !uv pip install ultralytics
3  from ultralytics import YOLO
4  import cv2
5  from google.colab.patches import cv2_imshow
6  ultralytics.checks()
7

```

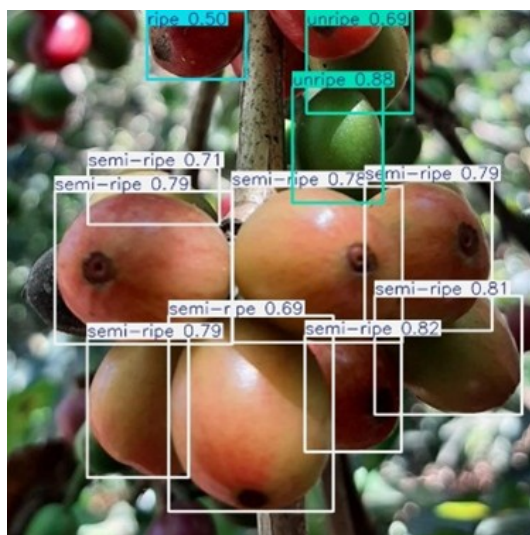
```

8      #Vincular Google Drive
9      from google.colab import drive
10     drive.mount('/content/drive')
11
12     # Cargar el modelo
13     model = YOLO('/content/drive/MyDrive/Colab Notebooks/
14                  prueba_15_yolov11m_v15(3)--/weights/best.pt')
15
16     # Prediccion del modelo
17     from PIL import Image
18     # Procesa y muestra una imagen
19     path = "/content/drive/MyDrive/Colab Notebooks/prueba_08.jpg"
20     results = model.predict(path, save=True, imgsz = 640, conf
21                             =0.5)
22
23     # Visualiza los resultados
24     for i, r in enumerate(results):
25         im_bgr = r.plot()
26         im_rgb = Image.fromarray(im_bgr[..., ::-1])
27
28         r.show()
29
30         r.save(filename=f"results{i}.jpg")

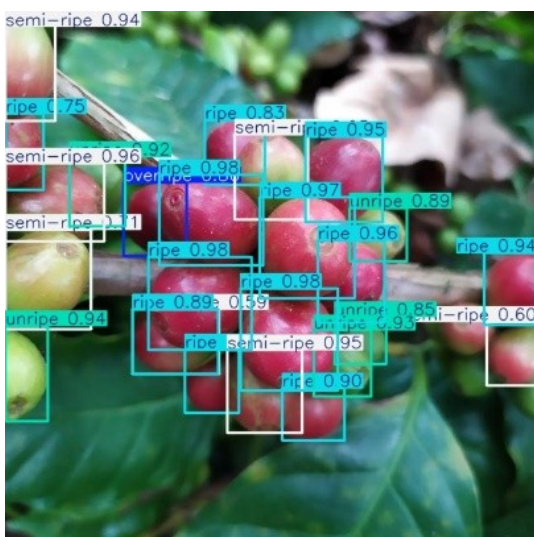
```



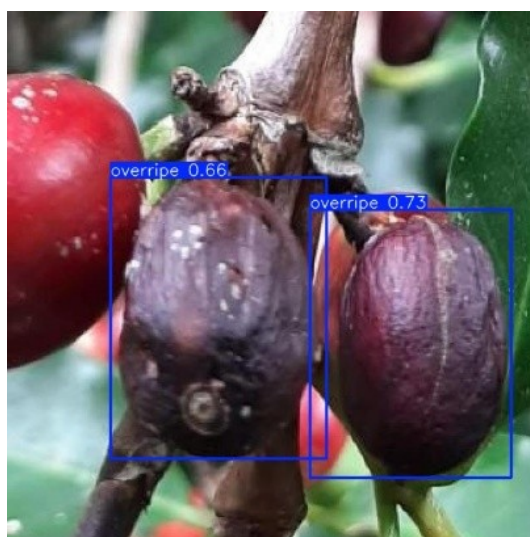
(a)



(b)



(c)



(d)



(e)

Figura 43: Visualización de la predicción del modelo YOLO11m: (a) Predicción en categoría unripe, (b) Predicción en categoría semi-ripe, (c) Predicción en categoría ripe, (d) Predicción en categoría overripe y (e) Predicción del modelo en video.

CAPÍTULO 4: IMPLEMENTACIÓN DEL MODELO DE RED NEURONAL CONVOLUCIONAL.

En este capítulo se describirán los pasos realizados para la implementación del modelo YOLO en un sistema de detección de madurez de cerezas de café.

En primera instancia, se pensó en implementar el modelo de detección de madurez de cerezas de café en un sistema embebido, específicamente en una Raspberry Pi 4B, donde se le diseñaría una carcasa impresa en 3D para ser un dispositivo portátil. Sin embargo, debido a cuestiones referentes a la cantidad de módulos adicionales que se le tendrían que conectar a la Raspberry Pi (cámara, batería, pantalla, entre otros), el tamaño y peso del dispositivo ensamblado y a la limitación de recursos computacionales que tiene, se optó por implementar un sistema basado en la nube y que se pudiera acceder desde una interfaz gráfica en un dispositivo móvil.

Dicho lo anterior, en la figura 44 se muestra un diagrama de bloques sobre el proceso de implementación del sistema clasificador de madurez de cerezas de café, el cual se divide en la sección de backend y frontend. Dentro del backend se encuentra la contenerización del modelo YOLO11m con la API REST y su despliegue en la nube, mientras que en el frontend se encuentra la aplicación móvil desarrollada en Flutter.

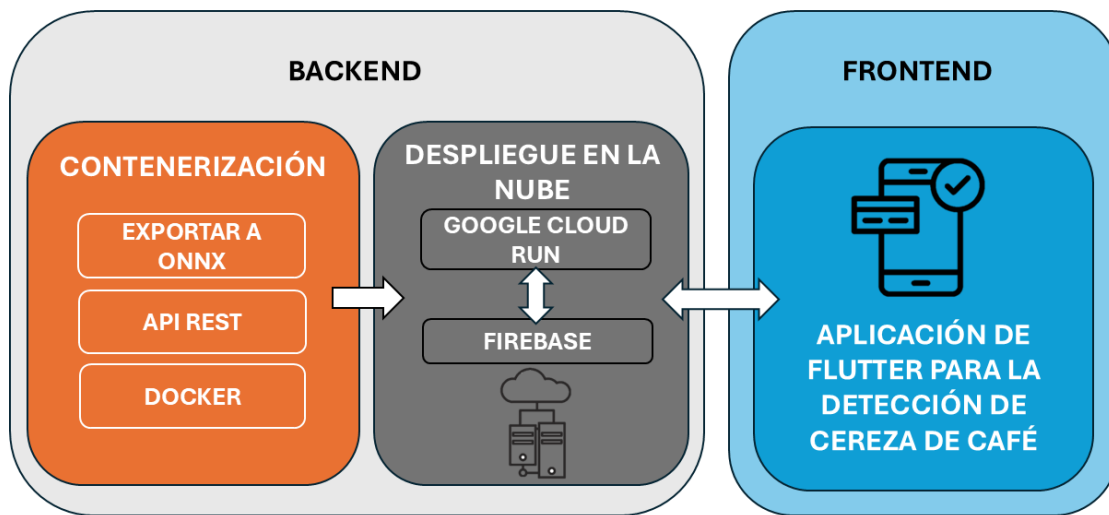


Figura 44: Diagrama de bloques de la implementación del sistema.

Este sistema permite la detección de madurez de cerezas de café mediante la toma de fotografías con un dispositivo móvil y el envío de estas a la API REST desplegada en la nube, la cual devuelve la imagen anotada con las detecciones de cerezas de café. Las herramientas utilizadas para la implementación del modelo YOLO11m fueron:

- Docker para la contenerización.
- Google Cloud Platform (Cloud Run y Artifact Registry) para el despliegue en la nube.
- Firebase para base de datos y almacenar las fotografías.
- Google Colab para la exportación del modelo.
- Visual Studio Code para la API REST.
- Flutter con Android Studio para la aplicación móvil.

4.1 BACKEND.

4.1.1 Contenerización.

4.1.1.1 Exportación del modelo

El modelo YOLO11m entrenado para la detección de cerezas de café fue exportado al formato ONNX (Open Neural Network Exchange), como se muestra en la Figura 45, utilizando el método de exportación de Ultralytics [81] con la plataforma de Google Colab.

```
from ultralytics import YOLO

# Load the YOLO11 model
model = YOLO("/content/drive/MyDrive/Colab Notebooks/prueba_15_yolov11m_v15(3)--/weights/best.pt")

# Export the model to ONNX format
model.export(format="onnx")
```

Figura 45: Código de exportación del modelo YOLO11m en formato ONNX.

Dicho formato genera una mayor aceleración del tiempo de ejecución del modelo en comparación con el formato nativo de PyTorch, logrando hasta un incremento de 3x en la velocidad de inferencia en CPU como lo comenta [82].

4.1.1.2 API REST

Con el fin de desplegar el modelo que se entrenó en el capítulo anterior 3 en un entorno real, se desarrolló una API REST, basado en el ejemplo que se muestra en la página web de Ultralytics [83], con el uso del editor de código Visual Studio Code. Se creó una carpeta llamada `detection-coffee-yolo`, que contiene el modelo y dos archivos de Python, `main` y `app`, cuyos códigos se encuentran de manera completa en los anexos 4.4.1 y 4.4.2. En el archivo `main` se utilizó el framework FastAPI, el cual expone dos endpoints, el primero, `/health`, verifica si el modelo está listo para recibir peticiones y el segundo, `/predict`, realiza la inferencia del modelo YOLO11m por medio del archivo `app`, obteniendo los resultados en un archivo JSON y una imagen anotada.

```

main.py x
src > main.py
85 async def root():
99 @app.post(AIP_PREDICT_ROUTE, response_model=PredictionResponse)
100 async def predict(request: PredictionRequest):
101     """Prediction endpoint."""
102     try:
103         predictions = [] # Lista para almacenar resultados de todas las instancias
104
105         # Procesa cada instancia en la request
106         for instance in request.instances:
107             if isinstance(instance, dict):
108                 if "image" in instance:
109                     # Decodifica la imagen de base64
110                     image_data = base64.b64decode(instance["image"])
111                     # Convierte bytes a imagen PIL/OpenCV
112                     input_image = get_image_from_bytes(image_data)
113                 else:
114                     raise HTTPException(status_code=400, detail="Instance must contain 'image' field")
115             else:
116                 raise HTTPException(status_code=400, detail="Invalid instance format")
117
118             # Extrae parámetros opcionales o usa valores por defecto
119             parameters = request.parameters or {}
120             confidence_threshold = parameters.get("confidence", 0.5)
121             return_annotated_image = parameters.get("return_annotated_image", True)
122
123             # Ejecuta inferencia con el modelo YOLO11
124             result = run_inference(input_image, confidence_threshold=confidence_threshold)
125             detections_list = result["detections"]
126
127             # Formatea las detecciones para la respuesta
128             detections = []
129             for detection in detections_list:
130                 formatted_detection = {
131                     "class_name": detection["name"],
132                     "confidence": detection["confidence"],
133                     "bbox": {
134                         "xmin": detection["xmin"],
135                         "ymin": detection["ymin"],
136                         "xmax": detection["xmax"],
137                         "ymax": detection["ymax"],
138                     },
139                 }
140                 detections.append(formatted_detection)
141
142             # Construye el resultado de predicción
143             prediction = {
144                 "detections": detections,
145                 "detection_count": len(detections),

```

Figura 46: Fragmento del código del archivo main.py.

Dentro del archivo `main` las librerías utilizadas fueron:

- `sys` para interactuar con el sistema
- `os` para variables de entorno
- `fastapi` framework para crear la API REST
- `pydantic` para validación de datos
- `typing` para anotaciones de tipos de datos

- `app` Archivo `app.py` que contiene la lógica de inferencia
- `uvicorn` servidor ASGI para ejecutar la aplicación FastAPI
- `base64` para codificación y decodificación de datos
- `loguru` para la visualización de logs

En el caso del archivo `app`, se utilizó el modelo YOLOm11 con el framework de ultralytics para realizar la detección de las cerezas de café, contabilizar las detecciones por clase, dibujar las cajas delimitadoras y devolver los resultados en un archivo JSON junto con una imagen anotada.

Dentro del archivo `app` las librerías utilizadas fueron:

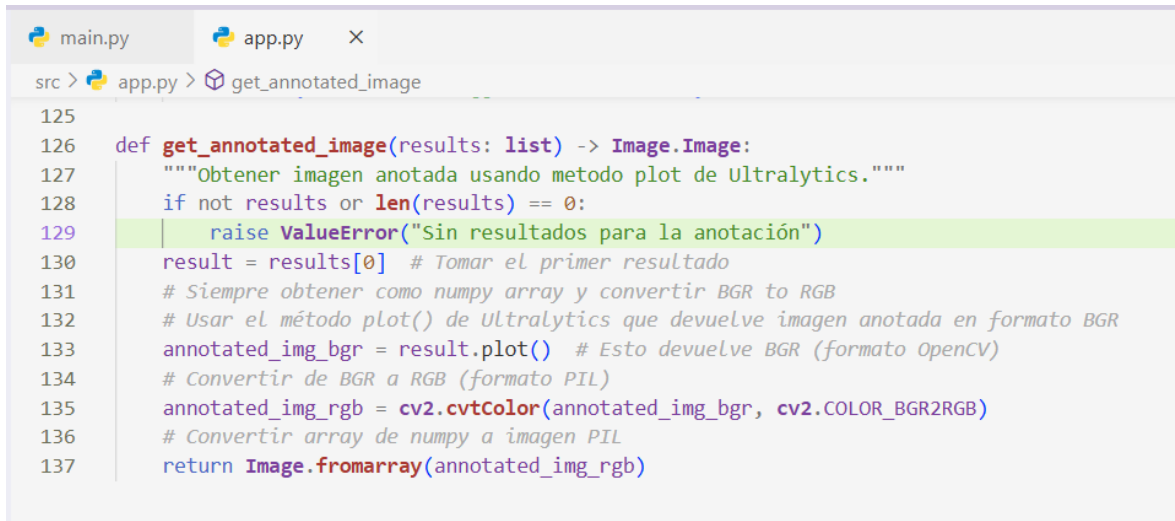
- `ultralytics` para cargar modelos y acceder a sus funciones.
- `PIL` para procesamiento de imágenes con Pillow.
- `io` para operaciones de entrada y salida.
- `os` para manejo de rutas de archivos y directorios.
- `typing` para anotaciones de tipos de datos.
- `cv2` para procesamiento de imágenes con OpenCV.
- `numpy` para operaciones numéricas con matrices.

Para iniciar la lógica de la detección de cerezas de café, se empezó con crear una función llamada `_initialize_model`, que carga el modelo YOLO11m personalizado y si no lo encuentra, utiliza un modelo YOLO11n pre-entrenado.

Después, se crearon otras dos funciones llamadas `get_image_from_bytes` y `get_bytes_from_image`, la primera convierte de bytes a imágenes PIL y la segunda convierte de imágenes PIL a bytes.

A continuación se generó una función para la inferencia llamada `run_inference`, donde se utiliza el método `predict` de ultralytics y procesa los resultados, extrayendo coordenadas de las cajas delimitadoras, confianzas y clases. Estos resultados se guardan en un archivo JSON.

Por último, la función `get_annotated_image`, como se muestra en la figura 47, utilizando la función `plot` de `ultralytics`, dibuja las cajas delimitadoras en la imagen y devuelve la imagen binaria anotada.

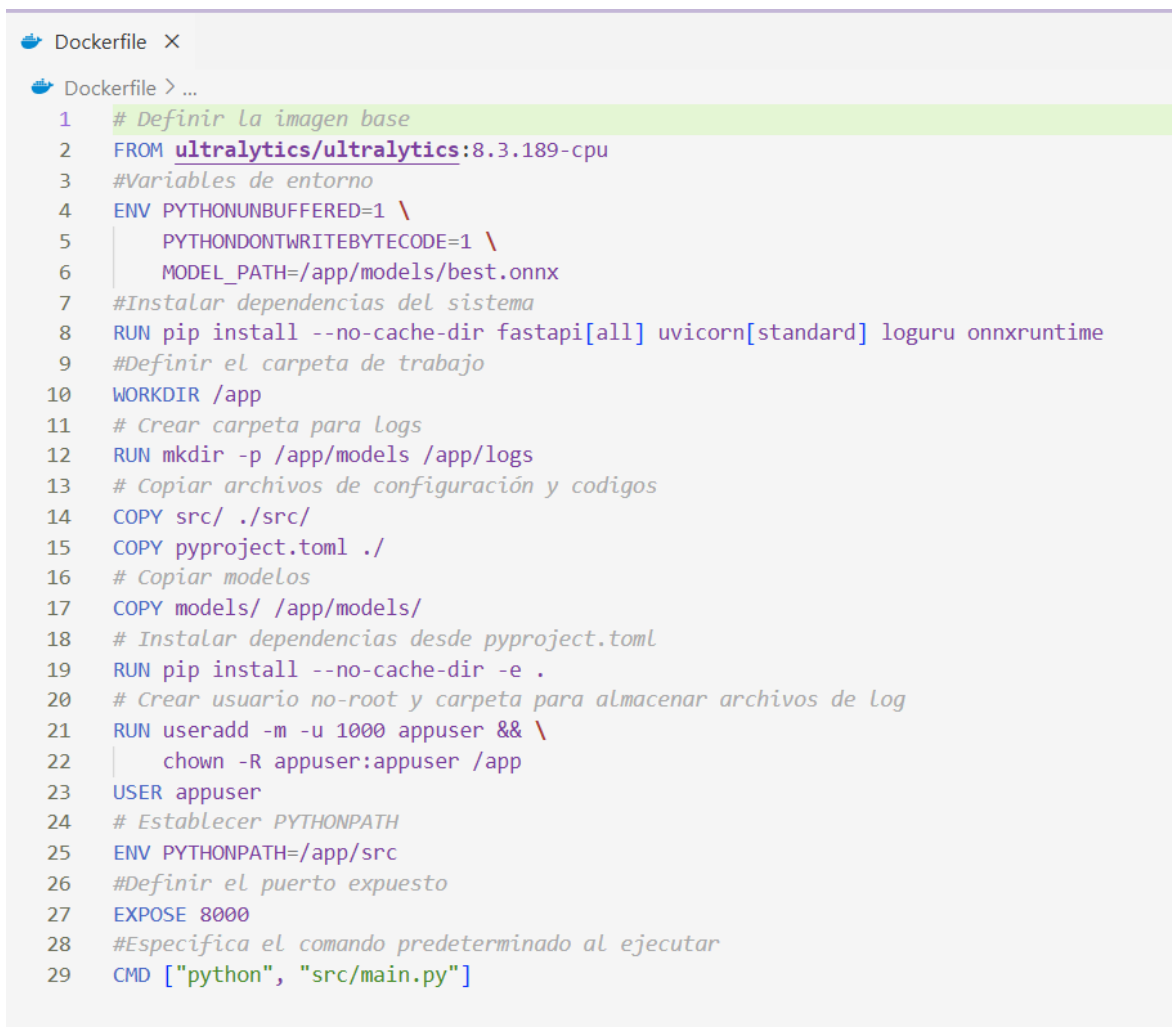


```
125
126 def get_annotated_image(results: list) -> Image.Image:
127     """Obtener imagen anotada usando metodo plot de Ultralytics."""
128     if not results or len(results) == 0:
129         raise ValueError("Sin resultados para la anotación")
130     result = results[0] # Tomar el primer resultado
131     # Siempre obtener como numpy array y convertir BGR to RGB
132     # Usar el método plot() de Ultralytics que devuelve imagen anotada en formato BGR
133     annotated_img_bgr = result.plot() # Esto devuelve BGR (formato OpenCV)
134     # Convertir de BGR a RGB (formato PIL)
135     annotated_img_rgb = cv2.cvtColor(annotated_img_bgr, cv2.COLOR_BGR2RGB)
136     # Convertir array de numpy a imagen PIL
137     return Image.fromarray(annotated_img_rgb)
```

Figura 47: Fragmento del código del archivo `app.py`.

4.1.1.3 Docker

Dentro de la misma carpeta `detection-coffee-yolo`, se creó un archivo `Dockerfile` para poder crear una imagen Docker que empaquetó la API REST, el modelo YOLO11m y un archivo `pyproject.toml` que especifica todas las librerías utilizadas en el servicio (FastAPI, Ultralytics, Pillow, Uvicorn y Loguru) para que se instalen de manera automática al crear la imagen del contenedor. La imagen base que se estableció en el `Dockerfile` fue la imagen oficial `ultralytics/ultralytics:8.3.189-cpu`, como se muestra en la figura 48.



```
Dockerfile X
Dockerfile > ...
1  # Definir la imagen base
2  FROM ultralytics/ultralytics:8.3.189-cpu
3  #Variables de entorno
4  ENV PYTHONUNBUFFERED=1 \
5      PYTHONDONTWRITEBYTECODE=1 \
6      MODEL_PATH=/app/models/best.onnx
7  #Instalar dependencias del sistema
8  RUN pip install --no-cache-dir fastapi[all] uvicorn[standard] loguru onnxruntime
9  #Definir el carpeta de trabajo
10 WORKDIR /app
11 # Crear carpeta para logs
12 RUN mkdir -p /app/models /app/logs
13 # Copiar archivos de configuración y codigos
14 COPY src/ ./src/
15 COPY pyproject.toml ./
16 # Copiar modelos
17 COPY models/ /app/models/
18 # Instalar dependencias desde pyproject.toml
19 RUN pip install --no-cache-dir -e .
20 # Crear usuario no-root y carpeta para almacenar archivos de log
21 RUN useradd -m -u 1000 appuser && \
22     chown -R appuser:appuser /app
23 USER appuser
24 # Establecer PYTHONPATH
25 ENV PYTHONPATH=/app/src
26 #Definir el puerto expuesto
27 EXPOSE 8000
28 #Especifica el comando predeterminado al ejecutar
29 CMD ["python", "src/main.py"]
```

Figura 48: Código del Dockerfile utilizado para crear la imagen.

Para la visualización de la creación de la imagen en la plataforma Docker Desktop, como se muestra en la figura 49, se ejecutó el siguiente comando en la terminal Powershell:

```
docker build --platform linux/amd64 -t detect-cherry:v1 .
```

Donde:

- `docker build` es el comando para construir una imagen mediante el Dockerfile.
- `-platform linux/amd64` especifica la plataforma para la cual se está construyendo la imagen.
- `-t detect-cherry:v1` asigna nombre y etiqueta de la version a la imagen.

- `.` indica que la ruta donde se encuentran los archivos necesarios para construir la imagen es el directorio actual.

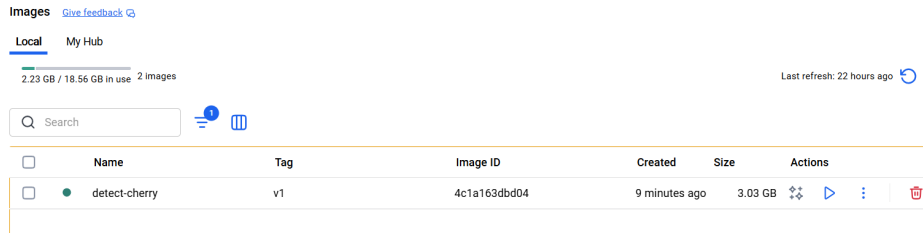


Figura 49: Imagen docker creada.

Para poder comprobar el funcionamiento de la imagen creada, se creó el contenedor ejecutando el siguiente comando en la misma terminal:

```
docker run -d -p 8081:8000 --name contenedor-cafe detect-cherry:v1
```

Donde:

- `docker run` es el comando para crear un contenedor a partir de una imagen.
- `-d` ejecuta el contenedor en segundo plano.
- `-p 8081:8000` son los puertos de publicación, 8081 es el número de puerto de la máquina host donde se desea recibir tráfico y 8000 es el número de puerto dentro del contenedor que está escuchando las conexiones.
- `--name contenedor-cafe` asigna un nombre al contenedor.
- `detect-cherry:v1` especifica el nombre y etiqueta de la imagen a usar para crear el contenedor.

Como se puede ver en la figura 50, al ejecutar la imagen Docker, el contenedor inició un servidor.

Containers [Give feedback](#)

Container CPU usage ⓘ
0.15% / 1200% (12 CPUs available)

Container memory usage ⓘ
173.9MB / 3.64GB

[Show charts](#)

Q Search  Only show running containers

☐	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
☐	contenedor-cafe	cb8e9d94ab34	detect-cherry-v1	8081:8000	0.15%	12 seconds ago	   

Figura 50: Contenedor docker en ejecución.

Con el servidor en ejecución, se realizó la inferencia del modelo, como muestra la figura 51, con el programa cURL.

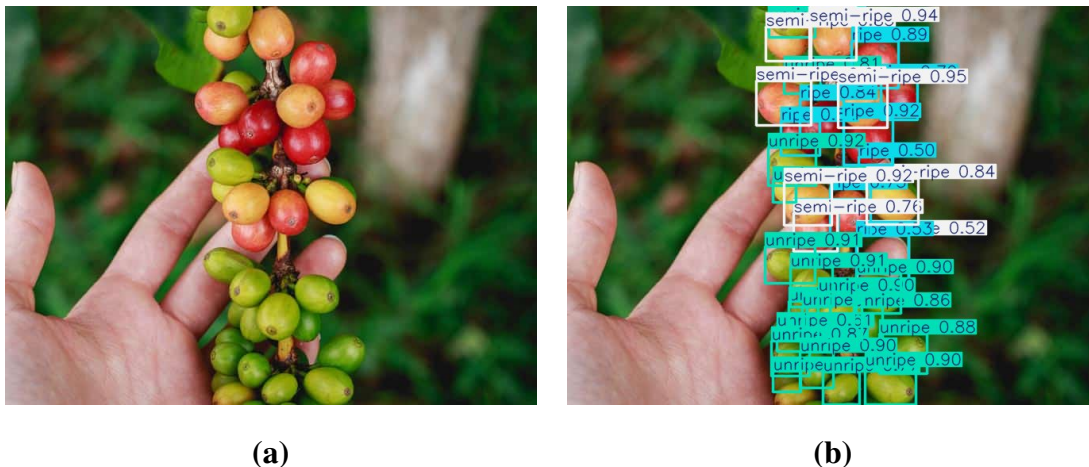


Figura 51: Inferencia del modelo con el programa cURL: (a) Imagen de prueba, (b) Resultado de inferencia.

Para ello, se utilizó un archivo JPG llamada `prueba.jpg` y se ejecutó el siguiente comando en otra ventana de la terminal Powershell dentro de la ubicación del archivo JPG:

```
curl.exe -X POST -F "image=@prueba.jpg"
http://localhost:8081/predict-image --output resultado.jpg
```

Donde:

- `curl.exe` es el comando para transferir datos desde o hacia un servidor.
- `-X POST` especifica que se realizará una solicitud HTTP POST.

- `-F image=@prueba.jpg"` indica que se enviará un formulario.
- `http://localhost:8081/predict-image` es la URL del endpoint de la API REST.
- `-output resultado.jpg` guarda la respuesta del servidor.

4.1.2 Despliegue en la nube.

4.1.2.1 Google Cloud Run

Con el modelo YOLO11m y la API REST empaquetados en una imagen Docker, se optó por un despliegue en la nube utilizando Google Cloud Platform, específicamente Cloud Run, gracias a su capacidad de escalabilidad automática (reduciendo a cero instancias en ausencia de tráfico para minimizar costos y escalando bajo demanda), en su integración directa con otros servicios de Google Cloud y en su compatibilidad con contenedores Docker, como lo menciona la documentación de Cloud Run [26].

Para iniciar, se ingresó a la plataforma web de Google Cloud con una cuenta de Google y se creó un nuevo proyecto, como se muestra en la figura 52.

Figura 52: Captura de pantalla de la creación de un nuevo proyecto en Google Cloud.

Una vez creado el proyecto, en este caso se le nombró como `prueba06`, se utiliza la terminal CMD de la máquina donde se está realizando la implementación para vincular Google Cloud con la imagen Docker utilizando el CLI (Interfaz de Línea de Comandos o Command Line Interface en inglés) de gcloud. Primero se activó el proceso de configuración inicial Google Cloud SDK, escribiendo en la terminal el siguiente comando:

```
gcloud init
```

A continuación, se obtuvo las credenciales de acceso para la cuenta de Google, escribiendo en la terminal el siguiente comando:

```
gcloud auth login
```

Después, se seleccionó el proyecto que se creó, escribiendo el ID del proyecto en la terminal el siguiente comando:

```
gcloud config set project prueba06-92e89
```

Posteriormente, se creó un repositorio privado en Artifact Registry para el almacenamiento de la imagen Docker como lo menciona la documentación de Google Cloud [84], que pudo ser creado desde la interfaz de la plataforma de Google Cloud o mediante el CLI de gcloud.

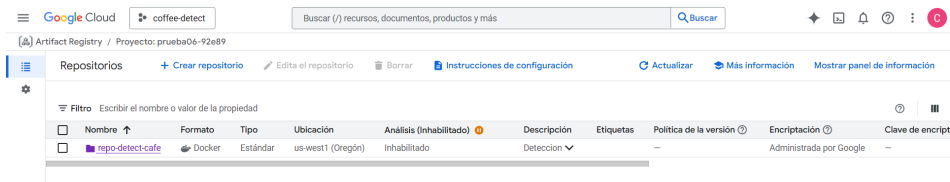


Figura 53: Visualización de repositorio en Artifact Registry.

En este caso se realizó por medio de la segunda opción y para poder visualizar la creación del repositorio en la plataforma web de Google Cloud como se observa en la Figura 53, ejecutando el siguiente comando en la terminal:

```
gcloud artifacts repositories create repo-detect-cafe
```

```
--repository-format=docker --location=us-west1
```

```
--description="Deteccion de cereza de cafe" --immutable-tags --async
```

Donde:

- `gcloud artifacts repositories create` es el comando para crear un repositorio en Artifact Registry.
- `repo-detect-cafe` es el nombre del repositorio.

- `-repository-format=docker` especifica el formato de la imagen del repositorio.
- `-location=us-west1` especifica la ubicación del repositorio, es Oregon, USA.
- `-description="Deteccion de cereza de cafe"` es la descripción de la función del repositorio.
- `-immutable-tags` significa que la imagen tendrá un tag, por lo que no podrá ser reemplazada.
- `-async` significa que tendrá una ejecución asíncrona, es decir, que puede seguir trabajando mientras se crea el repositorio.

A continuación, se ejecutó el siguiente comando para verificar que se haya creado el repositorio, como se puede ver en la Figura 54:

```
gcloud artifacts repositories list --project=prueba06-92e89
```

```
C:\Users\antho\StudioProjects\yolo_backend_functions>gcloud artifacts repositories create repo-detect-cafe --repository-format=docker --location=us-west1 --description="Deteccion de cereza de café" --immutable-tags --async
Create request issued for: [repo-detect-cafe]
Check operation [projects/prueba06-92e89/locations/us-west1/operations/5afd3eb8-f477-4b9c-9202-55a26bc94616] for status.

C:\Users\antho\StudioProjects\yolo_backend_functions>gcloud artifacts repositories list --project=prueba06-92e89
Listing items under project prueba06-92e89, across all locations.
```

REPOSITORY	FORMAT	MODE	DESCRIPTION	ARTIFACT_REGISTRY LOCATION	ARTIFACT_REGISTRY LABELS	ENCRYPTION	CREATE_TIME	UPDATE_TIME	SIZE (MB)
repo-detect-cafe	DOCKER	STANDARD_REPOSITORY	Deteccion de cereza de café	us-west1		Google-managed key	2025-10-15T14:34:13	2025-10-15T14:34:13	0

Figura 54: Salida de la terminal al crear el repositorio en Artifact Registry.

Después, se configuró la autenticación en el repositorio de Artifact Registry para Docker en la región us-west1, tal como dice la documentación de Google Cloud [85], ejecutando el siguiente comando:

```
gcloud auth configure-docker us-west1-docker.pkg.dev
```

Dentro de la sección de IAM y administración en la plataforma web de Google Cloud, como se muestra en la figura 55, se le agregó un rol llamado "Visualizador de objetos de Storage." al usuario que se está usando, para que tenga permiso de la creación de la imagen de Docker en el repositorio de Artifact Registry.

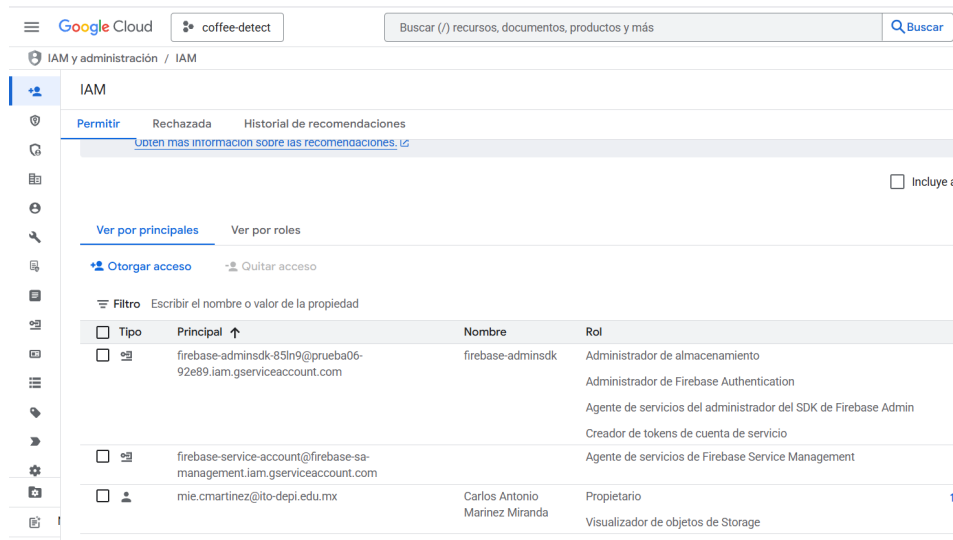


Figura 55: Visualización de los permisos en la sección IAM y administración en Google Cloud.

Una vez realizado lo anterior, se cambió el nombre y etiqueta de la imagen de Docker creada en la sub-sección anterior 4.1.1.3 para que pueda ingresar al repositorio de Artifact Registry, ejecutando el siguiente comando dentro de la ubicación donde se almacena la imagen de docker en la maquina local:

```
docker tag detect-cherry:v1
```

```
us-west1-docker.pkg.dev/prueba06-92e89/repo-detect-cafe/img-detect-cafe:v1
```

Donde:

- `docker tag` es el comando de Docker para etiquetar una imagen.
- `detect-cherry:v1` es el nombre y etiqueta de la imagen Docker previamente creada.
- `us-west1-docker.pkg.dev` es la region del repositorio de Docker con el nombre del host de Artifact Registry.
- `prueba06-92e89` es el ID del proyecto de Google Cloud.
- `repo-detect-cafe` es el nombre del repositorio.
- `img-detect-cafe:v1` es el nombre de la imagen y la etiqueta.

A continuación, se subió la imagen recién etiquetada al repositorio de Artifact Registry, ejecutando la siguiente línea de comando en la terminal dentro de la ubicación donde se encontraba almacenada la imagen de Docker en la maquina local:

```
docker push  
us-west1-docker.pkg.dev/prueba06-92e89/repo-detect-cafe/img-detect-cafe:v1
```

Donde:

- `docker push` es el comando de Docker para subir una imagen.
- `us-west1-docker.pkg.dev` es la region del repositorio de Docker con el nombre del host de Artifact Registry.
- `prueba06-92e89` es el ID del proyecto de Google Cloud.
- `repo-detect-cafe` es el nombre del repositorio.
- `img-detect-cafe:v1` es el nombre de la imagen y la etiqueta.

En la plataforma web de Google Cloud, se observó en la sección de Artifact Repository que aparece la imagen creada dentro del repositorio, como se muestra en la figura 56.

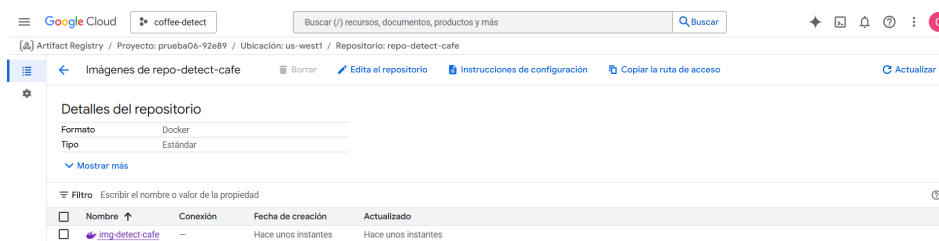


Figura 56: Visualización de la imagen Docker almacenada en el repositorio de Artifact Registry.

Finalmente, se ejecutó la siguiente línea de código en la terminal para crear el servicio con Cloud Run, como se muestra en la figura 57, con la imagen Docker que se almacenó en el repositorio de Artifact Registry:

Listing 4.1: desplegar imagen docker en Cloud Run

```

gcloud run deploy coffee-cherry-detector \
--image us-west1-docker.pkg.dev/prueba06-92e89/repo-detect-
    cafe/img-detect-cafe:v1 \
--region us-west1 \
--platform managed \
--memory 2Gi \
--cpu 1 \
--max-instances 5 \
--min-instances 0 \
--timeout 60s \
--concurrency 10 \
--port 8000 \
--allow-unauthenticated \
--set-env-vars="MODEL_PATH=/app/models/best.onnx" \
--set-env-vars="PYTHONPATH=/app/src"

```

Donde:

- `gcloud run deploy` es el comando de Google Cloud Run para crear un servicio.
- `coffee-cherry-detector` es el nombre del servicio.
- `-image us-west1-docker.pkg.dev/prueba06-92e89/repo-detect-cafe/img-detect-cafe:v1` es el nombre de la imagen
- `-region us-west1` es la region del servicio.
- `-platform managed` es la plataforma del servicio.
- `-memory 2Gi` es la memoria del servicio.
- `-cpu 1` es la cantidad de CPU del servicio.
- `-max-instances 5` es la cantidad maxima de instancias del servicio.

- `-min-instances 0` es la cantidad minima de instancias del servicio.
- `-timeout 60s` es el tiempo máximo de espera del servicio.
- `-concurrency 10` es la cantidad de instancias del servicio.
- `-port 8000` es el puerto del servicio.
- `-allow-unauthenticated` es la autenticación del servicio.
- `-set-env-vars="MODEL_PATH=/app/models/best.onnx"` es la variable de entorno del servicio.
- `-set-env-vars="PYTHONPATH=/app/src"` es la variable de entorno del servicio.

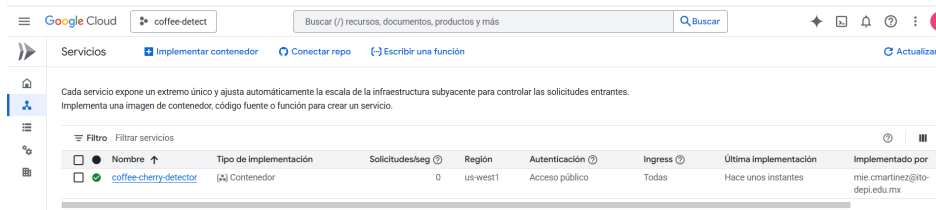


Figura 57: Visualización de la imagen Docker en función en Cloud Run.

4.1.2.2 Firebase

Con el fin de almacenar los datos que son proporcionados por la aplicación móvil (que se describirá en la sub-sección 4.2.1) y enviar la fotografía a Cloud Run, se empleó Firebase, específicamente Firestore Database, Firebase Storage, Firebase Authentication y Firebase Functions, cuya configuración fue por medio del CLI de Firebase y su plataforma web.

Para iniciar, usando la misma cuenta de Google que la que se usó en la plataforma web de Google Cloud, se ingresó a la plataforma web de Firebase, como se muestra en la figura 58, y se seleccionó el proyecto creado en la sección anterior 4.1.2.1.

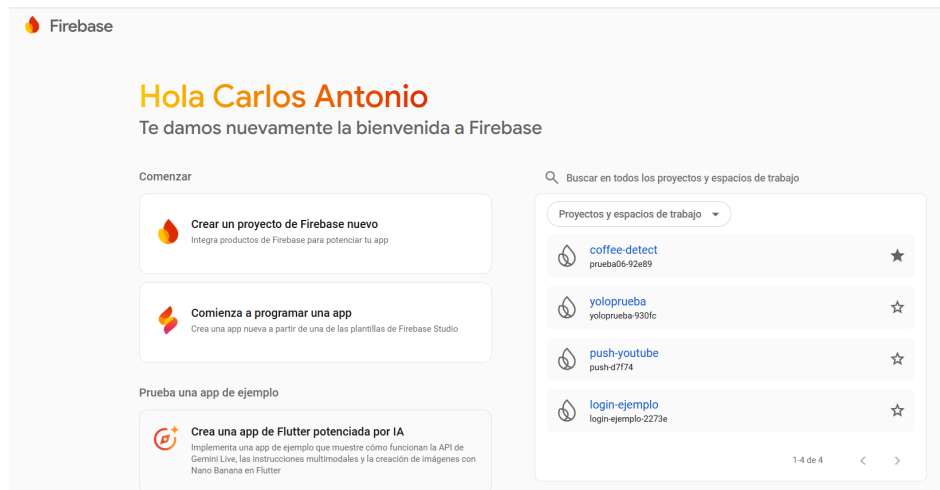


Figura 58: Captura de pantalla de la creación de un nuevo proyecto en Firebase.

Como parte inicial de la configuración de Firebase, se creó un proyecto nuevo de Flutter en Android Studio llamado `prueba06`, a pesar de que la aplicación móvil tiene una sección mas adelante explicando su funcionalidad, fue necesario crearlo en este punto para poder vincularlo con mayor facilidad a Firebase. En este punto, el proyecto nuevo de Flutter solo sirvió para instalar la dependencia `change_app_package_name`, que cambió el nombre del paquete de la aplicación en los archivos que lo contenían dentro del proyecto de Flutter de manera automática a `com.carlosmiranda.prueba06`, al ejecutar el siguiente comando en la terminal que proporciona Android Studio:

```
dart run change_app_package_name:main com.carlosmiranda.prueba06
```

Haciendo a un lado a Flutter por el momento, dentro de la terminal de CMD, se usó la CLI de Firebase, tal como se menciona en la documentación de Firebase [86], para iniciar sesión en Firebase con la cuenta de Google ejecutando el siguiente comando:

```
firebase login
```

Una vez que se inició sesión, se ejecutó el siguiente comando en la terminal para activar el CLI de FlutterFire, la cual permite configurar el proyecto de Flutter en Firebase:

```
dart pub global activate flutterfire_cli
```

A continuación, en la ubicación del proyecto de Flutter, se ejecutó el siguiente comando para

configurar Firebase, ingresando el ID del mismo proyecto que se utilizó en Google Cloud:

```
flutterfire configure --project=prueba06-92e89
```

Para que la aplicación de Flutter se ejecutara de manera correcta con Firebase, se le agregó al archivo `main.dart` del proyecto de flutter las siguientes líneas de código para importar el complemento principal de Firebase y el archivo de configuración que se generó en el paso anterior, además de inicializar Firebase con el objeto `DefaultFirebaseOptions`:

Listing 4.2: Inicializacion de Firebase

```
1  import 'package:firebase_core/firebase_core.dart';
2  import 'firebase_options.dart';
3
4  void main() async {
5      WidgetsFlutterBinding.ensureInitialized();
6      await Firebase.initializeApp(
7          options: DefaultFirebaseOptions.currentPlatform,
8      );
9      runApp(MyApp());
10 }
```

Después, se ingresó a la plataforma web de Firebase en su plataforma y se comprobó que estuviera conectado al proyecto de flutter, como se muestra en la figura 59.

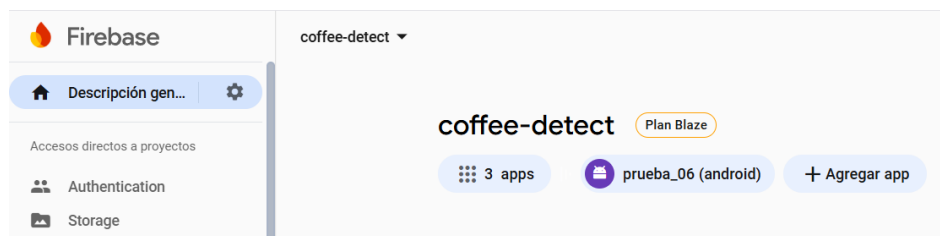


Figura 59: Visualización vinculación entre Firebase y Flutter.

A continuación, dentro de la plataforma web de Firebase, se ingresó a la sección de Firebase Authentication, Firestore Database y Firebase Storage para presionar el botón `Comenzar` para habilitar sus servicios, como se muestra en la figura 60.

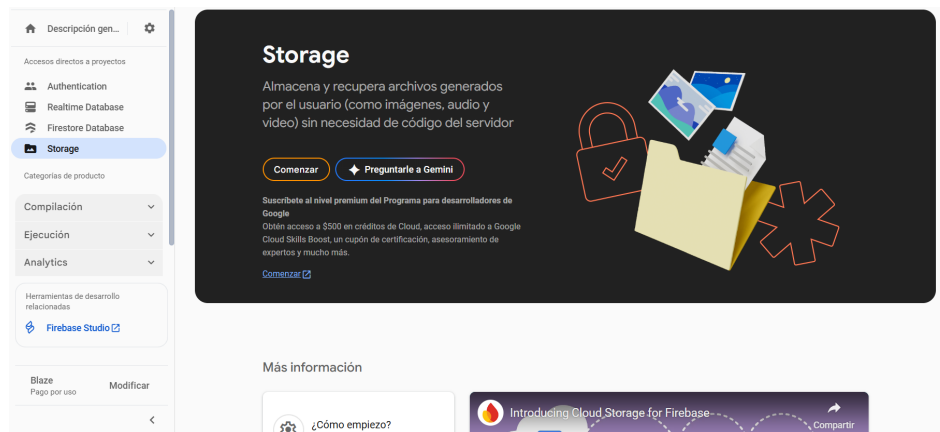


Figura 60: Visualización de la habilitación de Firebase Storage.

Por otro lado, el servicio de Firebase Functions fue utilizado para crear un ‘puente’ entre Firebase y Cloud Run. Como se hace mención en la documentación de Firebase [87], en una ubicación diferente a la carpeta donde se desarrolló la API REST, se creó una carpeta llamada `backend_functions`, la cual se vinculó con Firebase Functions haciendo uso del CLI de Firebase, instaladas mediante npm (Node Package Manager), con el siguiente comando en la terminal:

```
npm install -g firebase-tools
```

Dentro de la ubicación de la carpeta `backend_functions`, se ejecutó en la terminal el siguiente comando para inicializar el servicio de Firebase Functions:

```
firebase init functions
```

Se seleccionó el lenguaje de python para la creación del proyecto que se contendrá en la función de Firebase Functions. Lo siguiente, fue borrar el entorno virtual que Firebase genera por defecto para crear y activar uno nuevo manualmente.

Dentro de esta carpeta se escribió un código de Python llamado `main`, cuyo código completo se puede ver en el anexo 4.5.1, el cual envía la foto de manera automática desde Firebase Storage a Cloud Run, guarda la imagen anotada en Firebase Storage y registra los resultados en Firestore Database, para cada usuario.

Ademas, se creó un archivo de texto llamado `requirements` donde se especifica las librerías

utilizadas en el archivo `main`, las cuales son:

- `firebase_functions==0.4.3` para definir los eventos de Firebase Functions.
- `firebase-admin==6.8.0` para ingresar a Storage y Firestore.
- `requests>=2.28.1` para realizar peticiones HTTP.

Después se instaló manualmente las dependencias en el entorno virtual de Python, ejecutando los siguientes comandos en la terminal dentro de la ubicación de la carpeta:

```
pip install --upgrade pip
```

```
pip install -r requirements.txt
```

Con el código de python realizado y las dependencias instaladas, se implementó el archivo `main` en Firebase Functions ejecutando el siguiente comando en la terminal dentro de la ubicación de la carpeta, y posteriormente, se comprobó su implementación en la sección de Firebase Functions en la plataforma web de Firebase como se muestra en la figura 61.

```
firebase deploy --only functions
```

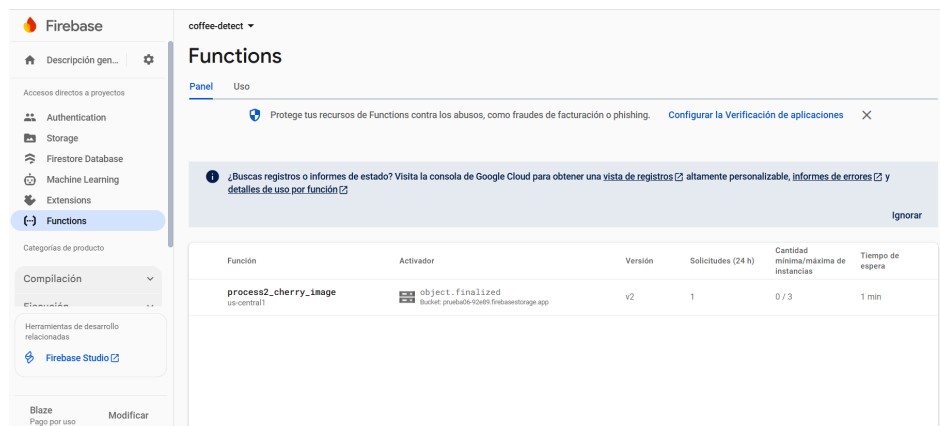


Figura 61: Visualización del panel de Firebase Functions

4.2 FRONTEND.

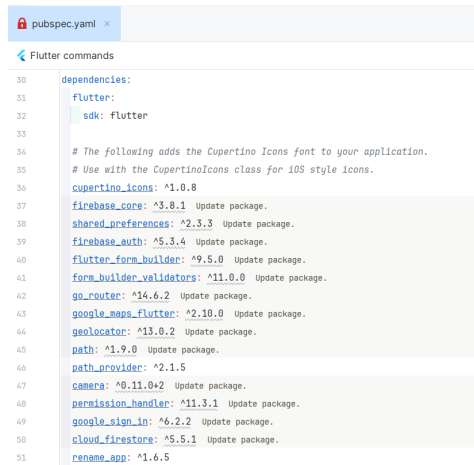
4.2.1 Aplicación móvil.

Para poder visualizar los resultados obtenidos en la detección de cerezas de café por el modelo YOLO11m en una interfaz gráfica, se desarrolló una aplicación móvil donde se recibe la imagen anotada para dispositivos con el sistema operativo de Android e incluso se puede ejecutar por la web, utilizando el lenguaje de programación Dart en conjunto con el framework Flutter usando el editor Android Studio.

Se optó por usar el lenguaje dart con el framework Flutter, debido a que permite desarrollar aplicaciones multiplataforma (Android, iOS, Web, Windows, MacOS y Linux) con una sola base de código, a diferencia del lenguaje de python, que a pesar de ser uno de los lenguajes más populares del mundo, no tiene un funcionamiento adecuado en dispositivos móviles.

Esta aplicación no se realizó para los sistemas iOS y macOS debido a que necesitan del entorno de desarrollo Xcode para compilar apps, el cual solo está disponible para productos de Apple.

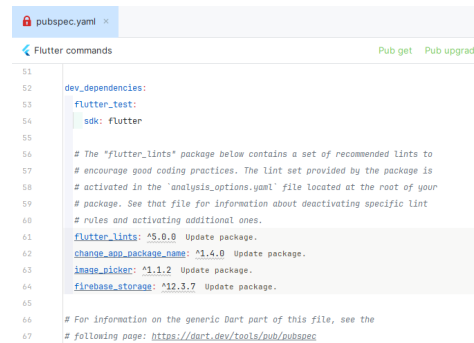
Para el desarrollo de la aplicación móvil, como se mencionó en la sección anterior [4.1.2.2](#), se creó un nuevo proyecto de Flutter en Android Studio llamado `prueba06`. Dentro de este proyecto, se instalaron las siguientes dependencias en el archivo `pubspec.yaml`, las cuales se muestran en la figura [62](#).



```
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  firebase_core: ^3.8.1 Update package.
  shared_preferences: ^2.3.3 Update package.
  firebase_auth: ^5.3.4 Update package.
  flutter_form_builder: ^9.5.0 Update package.
  form_builder_validators: ^11.0.0 Update package.
  go_router: ^14.6.2 Update package.
  google_maps_flutter: ^2.10.0 Update package.
  geolocator: ^13.0.2 Update package.
  path: ^1.9.0 Update package.
  path_provider: ^2.1.5
  camera: ^0.11.0+2 Update package.
  permission_handler: ^11.3.1 Update package.
  google_sign_in: ^6.2.2 Update package.
  cloud_firestore: ^5.5.1 Update package.
  rename_app: ^1.6.5
```

(a)



```
dev_dependencies:
  flutter_test:
    sdk: flutter

  # The 'flutter_lints' package below contains a set of recommended lints to
  # encourage good coding practices. The lint set provided by the package is
  # activated in the 'analysis_options.yaml' file located at the root of your
  # package. See that file for information about deactivating specific lint
  # rules and activating additional ones.
  flutter_lints: ^5.0.0 Update package.
  change_app_package_name: ^1.4.0 Update package.
  image_picker: ^1.1.2 Update package.
  firebase_storage: ^12.3.7 Update package.

# For information on the generic Dart part of this file, see the
# following page: https://dart.dev/tools/pub/pubspec
```

(b)

Figura 62: Captura de pantalla de las dependencias del proyecto: (a) dependencias del proyecto Flutter y (b) dependencias de desarrollo del proyecto Flutter.

Además se configuró el archivo `android/app/build.gradle`, como se muestra en la figura 63, el cual permite que funcione de manera correcta la aplicación móvil en dispositivos Android. En este se estableció el valor de `minSdk` a 23, esto quiere decir que, el nivel mínimo de API que se requiere para ejecutar esta aplicación móvil será 23, lo que es igual a la version de Android 6.0. La tabla de nivel de API se puede observar en la documentación oficial [88]. También se estableció el valor de `ndkVersion` a 25.1.8937393, que es la versión del conjunto de herramientas que permite usar código C y C++ con Android para administrar actividades nativas y acceder a componentes de dispositivos físicos, como sensores y entrada táctil, como lo dice la documentación oficial de Android [89].

```

11  android
12      namespace = "com.carlosmiranda.prueba06"
13      compileSdk = flutter.compileSdkVersion
14      ndkVersion = "25.1.8937393"
15
16      compileOptions {
17          // Enable core library desugaring to support Java 8+ language features
18          coreLibraryDesugaringEnabled true
19
20          sourceCompatibility = JavaVersion.VERSION_11
21          targetCompatibility = JavaVersion.VERSION_11
22      }
23
24      kotlinOptions {
25          jvmTarget = JavaVersion.VERSION_11
26      }
27
28      defaultConfig {
29          // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id.html).
30          applicationId = "com.carlosmiranda.prueba06"
31          // You can update the following values to match your application needs.
32          // For more information, see: https://flutter.dev/to/review-gradle-config.
33          minSdk = 23
34          targetSdk = flutter.targetSdkVersion
35          versionCode = flutter.versionCode
36          versionName = flutter.versionName
37          multiDexEnabled true
38      }

```

Figura 63: Captura de pantalla de la configuración del archivo build.gradle.

El flujo de trabajo de la aplicación móvil es tal como se muestra en la figura 64.

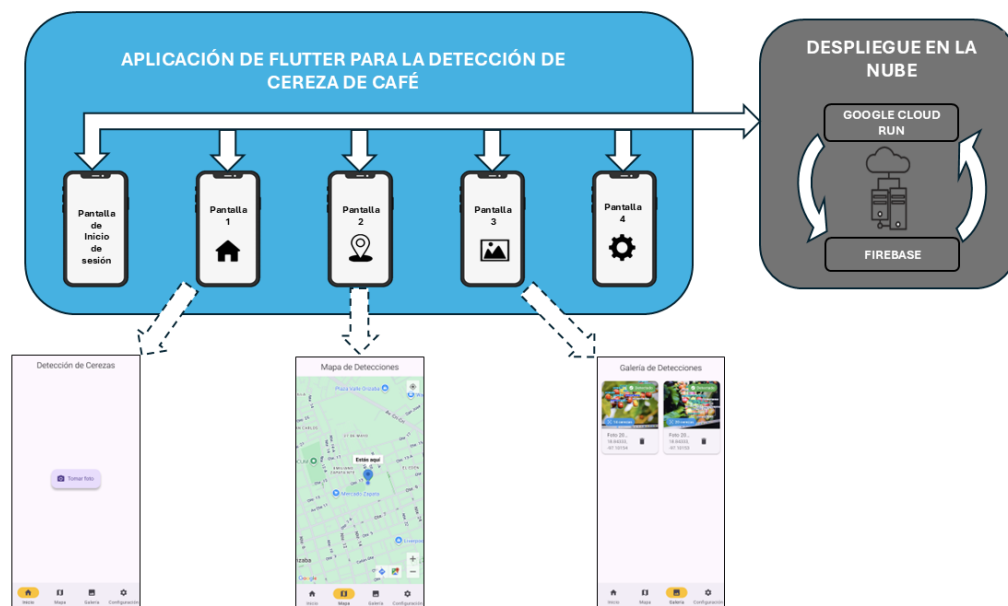


Figura 64: Diagrama del flujo de trabajo de la aplicación móvil.

Donde se ingresa a la aplicación móvil donde se muestra una pantalla de inicio, en el cual el usuario podrá iniciar sesión o crear una cuenta, como se muestra en la figura 65, esto de manera necesaria y obligatoria para ser vinculado con Firebase Authentication y así, poder ingresar a las siguientes pantallas de la aplicación. La razón por la que se implementó este método de autenticación es para que cada usuario tenga su propia base de datos de las detecciones de cereza de café.

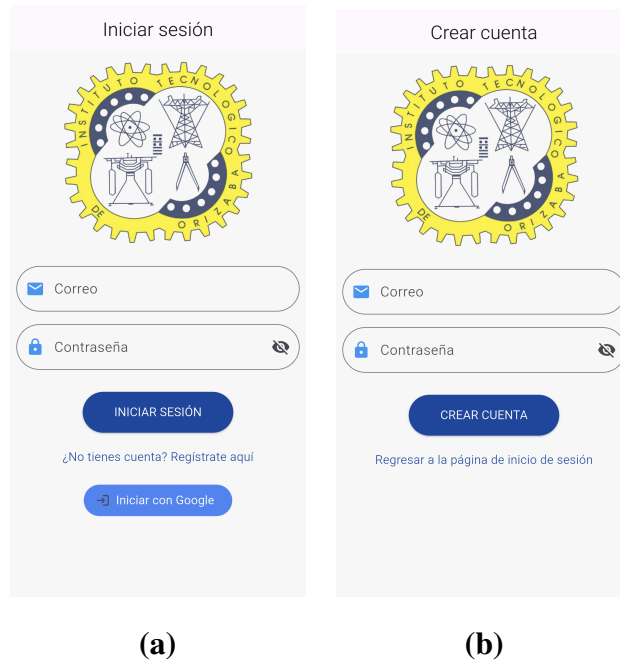


Figura 65: Pantallas para ingresar a la aplicación móvil: (a) pantalla de inicio de sesión y (b) pantalla para crear cuenta.

A continuación, como se muestra en la figura 66, la aplicación móvil tiene una barra de navegación que permite acceder a 4 pantallas diferentes. La primera es la pantalla principal donde el usuario puede capturar una imagen con la cámara de su dispositivo móvil, la segunda es un mapa donde el usuario puede ver la ubicación de las imágenes que se han capturado, la tercera es una galería de imágenes con las detecciones realizadas por el modelo y la cuarta es un perfil donde el usuario puede ver información sobre la aplicación y cerrar sesión.

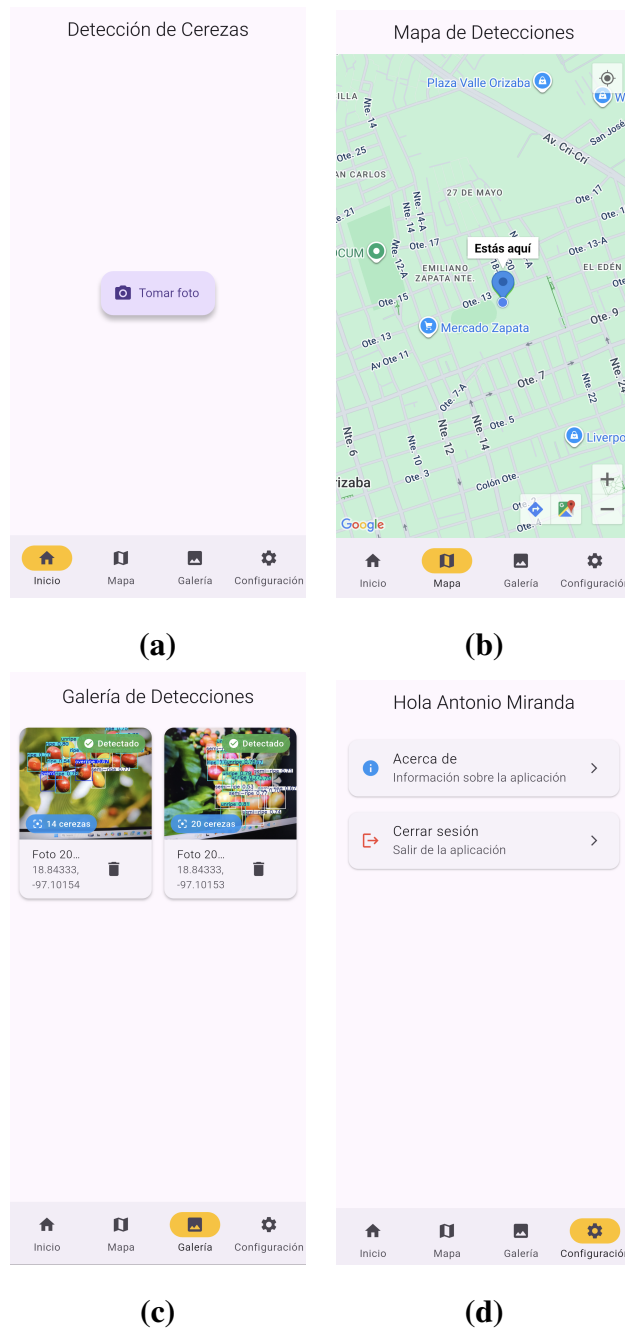


Figura 66: Pantallas de la aplicación móvil: (a) pantalla inicial para tomar fotografía, (b) pantalla del mapa, (c) pantalla de galería, (d) pantalla de configuración.

Dentro de la primera pantalla, los usuarios capturan imágenes desde sus dispositivos móviles, como muestra la figura 67 y luego se envían estas a Firebase Storage y la información de las imágenes y del usuario a Firestore Database.

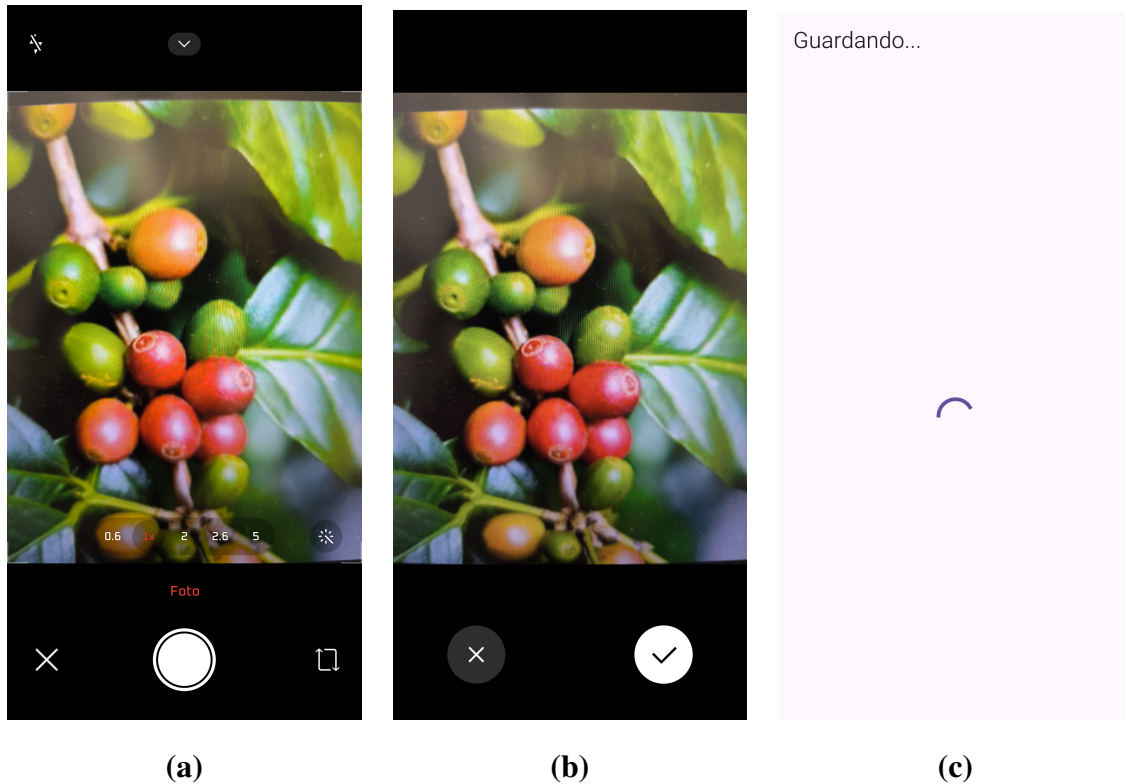


Figura 67: Captura de pantalla al presionar botón de tomar fotografía: (a) antes de tomar fotografía, (b) después de tomar fotografía y (c) subiendo imagen a Storage.

En Storage, cada imagen se almacena siguiendo esta estructura:

Ruta: `user/userID/photos/imageID/`

Donde `userID` es el identificador único del usuario y `imageID` es el identificador único de cada imagen.

En Firestore, la información se organiza en la siguiente estructura de colecciones y documentos:

Ruta: `user/userID/photos/imageID/`

Cada documento en `userID` contiene los siguientes campos:

- **createdAt:** Fecha y hora de creación del usuario.
- **displayName:** Nombre del usuario.
- **email:** Correo electrónico del usuario.

- **lastLogin:** Fecha y hora del último inicio de sesión.
- **photoURL:** URL de la foto de perfil del usuario.
- **provider:** Proveedor de autenticación (por ejemplo, correo electrónico).
- **uid:** Identificador único del usuario.

Dentro de cada documento en `imageID` se encuentran los siguientes campos:

- **attempts:** Número de intentos de Captura.
- **class_counts:** Conteo de cerezas detectadas por clase.
- **createdAt:** Fecha y hora de creación de la foto.
- **detection_count:** Número total de cerezas detectadas.
- **detection_status:** Estado de la detección (pendiente, en proceso, completada).
- **detection:** Ubicación de las cerezas detectadas en la imagen.
- **image_url:** URL de la imagen original almacenada en Storage.
- **lat:** Latitud de la ubicación donde se tomó la foto.
- **lon:** Longitud de la ubicación donde se tomó la foto.
- **name:** Nombre de la imagen.
- **original_image_path:** Ruta de la imagen original en Storage.
- **processed_at:** Fecha y hora de procesamiento de la imagen.
- **processed_image_path:** Ruta de la imagen procesada en Storage.
- **processed_image_url:** URL de la imagen procesada almacenada en Storage.
- **userId:** Identificador único del usuario.

Al momento de enviar la imagen a Storage, se activa Firebase Functions, que envía la imagen almacenada en Storage al modelo desplegado en Google Cloud Run, dicha funcionalidad y configuración se explicó en la subsubsección pasada [4.1.2.2](#).

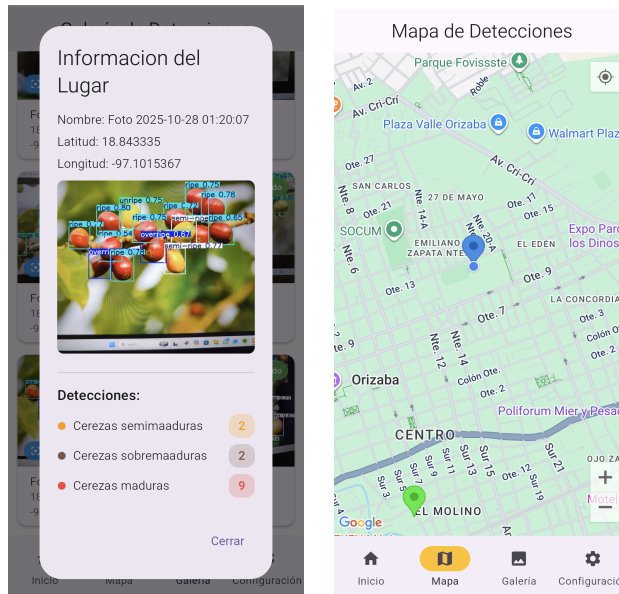
Una vez que el modelo haya realizado la inferencia, la imagen anotada se alojará en una nueva carpeta de Storage llamada `resultados`. La estructura de carpetas para las imágenes devueltas por la API REST es la siguiente:

Ruta: `resultados/userID/photos/imageID/`

Donde `userID` es el identificador único del usuario y `imageID` es el identificador único de cada imagen.

En Firestore, se actualizarán los campos que estén relacionados con la información sobre los resultados obtenidos en la inferencia (`detection_count`, `detection_status`, `detection`, `class_counts`, `processed_image_url`, `processed_image_path` y `processed_at`), dentro de los documentos `userID` y `imageID`.

Finalmente, la aplicación móvil mostrará en la tercera pantalla la imagen procesada con las detecciones y en la segunda pantalla la ubicación de la imagen en el mapa, como se muestra en la figura [68](#).



(a)

(b)

Figura 68: Resultados visualizados en la aplicación móvil: (a) información sobre las detecciones en la imagen y (b) ubicación de las imágenes en el mapa.

Con el flujo de trabajo de la aplicación explicado, a continuación se describirá la estructura del proyecto de Flutter, que se encuentra en una carpeta llamada `lib`, la cual contiene 3 archivos de lenguaje de programación Dart y 4 carpetas que, para su mejor comprensión, se pueden observar fragmentos de los códigos pertenecientes a este proyecto en la sección de anexos 4.3 (No se muestran los códigos completos, ya que se encuentra en proceso el registro de la aplicación móvil).

```
prueba06/
├── lib/
│   ├── main.dart
│   ├── firebase_options.dart
│   ├── navigation_bar_widget.dart
│   ├── services/
│   │   ├── storage_service.dart
│   │   └── auth.dart
│   └── pages/
```



```

├── login_page.dart
├── register_page.dart
├── preferences/
│   └── pref_usuarios.dart
├── pantallas/
│   ├── Pantalla_00/
│   │   ├── pantalla_00.dart
│   │   ├── camera_screen.dart
│   │   └── picture_screen.dart
│   ├── Pantalla_01/
│   │   ├── pantalla_01.dart
│   │   └── mapa.dart
│   ├── Pantalla_02/
│   │   ├── pantalla_02.dart
│   │   ├── place_draft.dart
│   │   ├── place.dart
│   │   ├── manage_place.dart
│   │   └── place_dialog.dart
│   └── Pantalla_03/
│       ├── pantalla_03.dart
│       └── informacion.dart
└── pubspec.yaml

```

Los archivos dentro de la carpeta `lib` son:

- El archivo `main` donde se inicializa la aplicación y se realiza la configuración de la navegación entre pantallas, como se muestra en el anexo [4.3.1](#).
- El archivo `firebase_options` donde se muestra la configuración de la vinculación entre el proyecto de Flutter y Firebase, el cual se genera de manera automática al introducir los comandos vistos en la subsubsección pasada [4.1.2.2](#).

- El archivo `navigation_bar_widget` donde se muestra la configuración de la visualización de las pantallas, como se muestra en el anexo [4.3.2](#).

En el caso de las cuatro carpetas dentro de la carpeta `lib`, se encuentran:

- La carpeta `pages` donde se muestra la lógica y visualización de la pantalla de inicio de sesión con el archivo de dart `login_page` y de la pantalla de creación de cuenta con el archivo de dart `register_page`, como se muestra en el anexo [4.3.3](#) y en el anexo [4.3.4](#).
- La carpeta `pantallas`, el cual contiene a su vez cuatro carpetas donde contienen la lógica para funcionalidad de las pantallas de la aplicación.
- La carpeta `preferences` donde se realiza la configuración de las preferencias del usuario con el archivo de dart `pref_usuarios`, como se muestra en el anexo [4.3.5](#).
- La carpeta `services` donde se realiza la lógica para guardar los usuarios en Firebase con el archivo de dart `auth` y donde se realiza la lógica para subir la imagen a Storage con el archivo de dart `storage_services`, como se muestra en el anexo [4.3.6](#) y en el anexo [4.3.7](#).

Dentro de la carpeta llamada `pantallas` se encuentran las siguientes carpetas:

- La carpeta `Pantalla_00` contiene tres archivos de dart donde, `pantalla_00` crea el botón para capturar la imagen, `camera_screen` realiza la lógica para capturar la imagen y `picture_screen` envía la imagen con los datos a Firebase y guarda la ubicación geográfica actual del usuario en una base de datos local, como se pueden observar en los anexos [4.3.8](#), [4.3.9](#) y [4.3.10](#).
- La carpeta `Pantalla_01` contiene dos archivos de dart donde, `pantalla_01` configura la pantalla para el mapa de ubicación y `mapa` carga el mapa de Google Maps con los marcadores para el usuario y las imágenes que se están capturando, como se muestra en el anexo [4.3.11](#) y [4.3.12](#).

- La carpeta `Pantalla_02` contiene cinco archivos de dart donde, `pantalla_02` configura la pantalla para almacenar las imágenes capturadas y los archivos (`place_draft`, `manage_place`, `place` y `place_dialog`) se encargan de la lógica para realizar una base de datos que contenga la ubicación de las imágenes capturadas y mostrar esta información junto con la imagen, cuyos fragmentos de códigos se muestran en los anexos [4.3.13](#), [4.3.14](#), [4.3.15](#), [4.3.16](#) y [4.3.17](#).
- La carpeta `Pantalla_03` contiene dos archivos de dart donde, `pantalla_03` crea los botones para salir de la sesión del usuario y para proporcionar información sobre la aplicación e `información` proporciona información sobre la aplicación, como se muestra en el anexo [4.3.18](#) y el anexo [4.3.19](#).

Para finalizar este capítulo, la aplicación móvil tuvo que ser probado en un dispositivo móvil sin estar conectado a la computadora local, por lo que creó un APK y siguiendo los pasos de la documentación oficial de *Build and release an Android app* [90], se ejecutó el siguiente comando en la terminal dentro de la ubicación del proyecto de Flutter:

```
flutter build apk --release
```

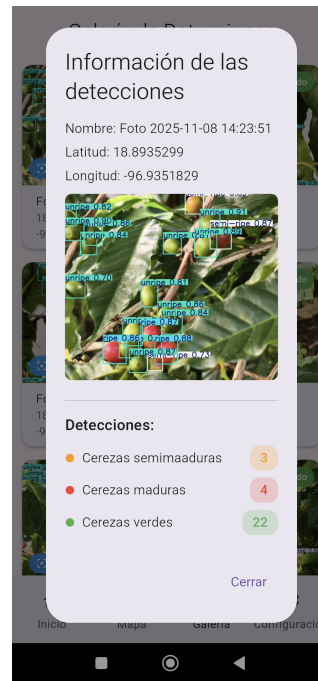
Donde el archivo final, `app-release.apk`, se mostró en la siguiente ubicación para después ser copiado al dispositivo móvil y de esta manera, usar la versión finalizada en cualquier dispositivo móvil que sea compatible:

```
build/app/outputs/flutter-apk/app-release.apk
```

Después, se realizó la prueba del sistema clasificador en los cafetos que se encuentran en el museo del café de Córdoba como se muestran en las figuras [69](#) y [70](#).



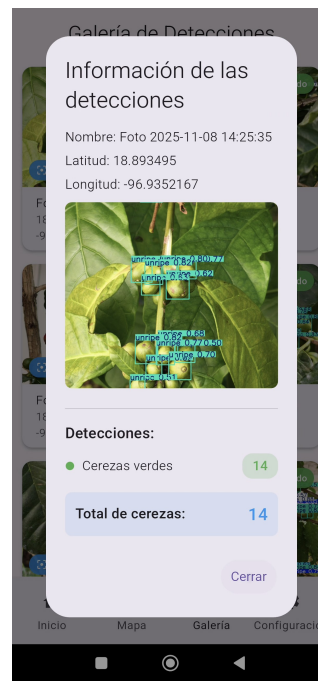
(a)



(b)



(c)

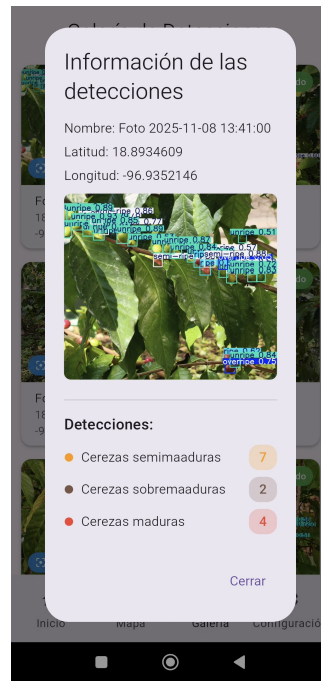


(d)

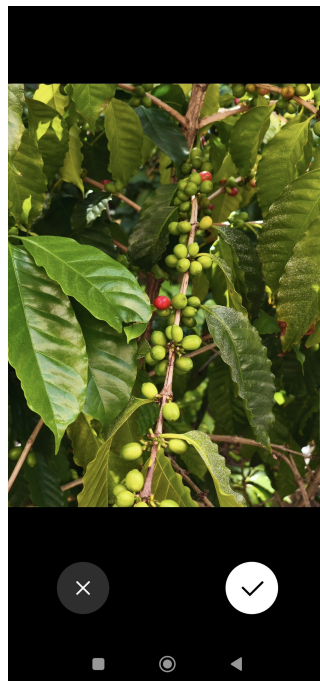
Figura 69: Resultados de la prueba en campo visualizados en la aplicación móvil: (a) captura de pantalla de la fotografía, (b) información sobre las detecciones en la fotografía, (c) captura de pantalla de la fotografía, (d) información sobre las detecciones en la fotografía.



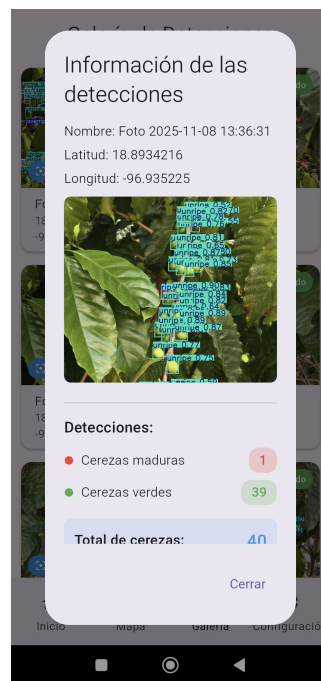
(a)



(b)



(a)



(b)

Figura 70: Resultados de la prueba en campo visualizados en la aplicación móvil: (a) captura de pantalla de la fotografía, (b) información sobre las detecciones en la fotografía, (c) captura de pantalla de la fotografía, (d) información sobre las detecciones en la fotografía.

CONCLUSIONES

La presente investigación, la cual se enfocó en desarrollar e implementar un sistema clasificador de color de las cerezas de café para determinar su madurez mediante técnicas de procesamiento de imágenes e inteligencia artificial, cumplió con los objetivos propuestos.

Uno de estos objetivos fue el entrenamiento de una red neuronal convolucional para realizar la función de clasificación del sistema, donde se utilizó una arquitectura YOLO11 en su tamaño intermedio (YOLO11m). Primero se adquirió y etiquetó un conjunto de datos para que posteriormente se realizara el entrenamiento de un modelo que detectara y clasificara las cuatro clases de maduración (verde, semi-maduro, maduro y sobre-maduro) de las cerezas de café mediante el lenguaje de programación Python. Al evaluar el modelo resultante en el subconjunto de pruebas, mostró un desempeño satisfactorio, ya que se obtuvo un valor general de 0.864 en la métrica mAP@50 y, de manera individual, se obtuvieron valores superiores a 0.817 de la misma métrica en las cuatro clases, siendo la clase verde la que mejor identificada (0.905) y la sobre-madura la más difícil (0.817), debido a la menor cantidad de etiquetas en el conjunto de datos para esta clase.

Otro objetivo propuesto fue la realización del despliegue del modelo obtenido después del entrenamiento, el cual se realizó por medio de la nube, específicamente en Google Cloud Platform. En este punto se implementó una API REST que, en conjunto con el modelo, realizó las detecciones y clasificaciones en imágenes que contuvieran cerezas de café, para que posteriormente fueran empaquetados en un contenedor Docker y almacenados en un servicio de Google Cloud Run. A pesar de ser una muy interesante y popular alternativa de despliegue para el modelo YOLO11m, debe tomarse en cuenta la dependencia de este con la conexión a internet y el [precio](#) por el uso de los servicios de Google Cloud Run, donde el precio por uso para Cloud Run en este caso es de \$0.15 pesos mexicanos al mes.

La elaboración de la interfaz gráfica para visualizar los resultados de la inferencia del modelo fue el último objetivo descrito y se optó por desarrollar una aplicación móvil en Flutter, cuya función es tomar las fotografías de las cerezas de café directamente en los plantíos de café y en esta misma, mostrar los resultados de las detecciones y clasificaciones que realizó el modelo

almacenado en Cloud Run. Para lograrlo, se utilizó Firebase Firestore para almacenar las imágenes capturadas por la aplicación y las imágenes con detecciones, Firebase Storage para almacenar información de los resultados de la inferencia y Firebase Functions para mandar y recibir las imágenes a Cloud Run. Al igual que en el objetivo anterior, se debe tener en cuenta el [precio](#) por el uso de los servicios de Firebase.

Al evaluar el sistema clasificador, se pudo observar que el modelo YOLO11m tuvo dificultad para detectar cerezas de café de tamaño pequeño, lo que implica la necesidad de tomar las fotografías a una distancia cercana para garantizar la identificación. Asimismo, la inferencia se vio afectada en la identificación de los frutos que están en zonas de sombra y de los que se encuentran en zonas con altos niveles de brillo debido a la luz solar.

A pesar de dichas limitaciones, el sistema clasificador de cereza de café se presenta como una herramienta práctica y accesible para los pequeños productores al momento de realizar el corte de la fruta durante la etapa de la cosecha. De esta manera se lleva la precisión observada en prácticas realizadas en entornos controlados a condiciones más demandantes como en fincas, con el potencial de mejorar la calidad final del café mexicano.

RECOMENDACIONES

Con la finalidad de mejorar la investigación presentada, se hacen las siguientes recomendaciones:

- Elaborar u obtener un conjunto de datos mas grande, enfocándose en elevar el número de imágenes donde aparezca la clase sobre-maduro y agregar imágenes con diferentes tipos de iluminación o sombreado e imágenes alejadas donde las cerezas de café sean objetos pequeños.
- Ampliar el número de clases de madurez, como lo menciona Marín-López et al. [8] en su artículo.
- Modificar los hiperparámetros del entrenamiento de la red neuronal convolucional para obtener un mejor modelo.
- Emplear versionas de YOLO más recientes como YOLO26 que en la fecha de realización de esta investigación, aun se encuentra en desarrollo.
- Mejorar el uso de recursos para el despliegue del modelo YOLO en la nube.
- Implementar el modelo YOLO dentro de la aplicación móvil para emigrar el despliegue en la nube al despliegue al borde y así evitar la dependencia de la conexión a internet.
- Escalar el sistema clasificador para futuras aplicaciones en grandes empresas productoras de café.

REFERENCIAS BIBLIOGRÁFICAS

- [1] Gobierno de México, *Café mexicano, a toda hora*, <https://www.gob.mx/agricultura/es/articulos/cafe-mexicano-a-toda-hora>, Accessed: 2025-10-02, 2017.
- [2] International Coffee Organization, *Información del mercado público*, <https://ico.org/es/resources/public-market-information/>, Accessed: 2025-10-02, 2024.
- [3] El café de México, *Nuestra historia*, <http://www.cafedemexico.mx/nuestra-historia.html>, Accessed: 2025-10-02.
- [4] J. Arcila-Pulgarín, L. Buhr, H. Bleiholder, H. Hack y H. Wicke, «Aplicación de la escala BBCH ampliada para la descripción de las fases fenológicas del desarrollo de la planta de café /Coffea/ sp.,» *Boletín Técnico Cenicafé*, vol. 23, págs. 1-32, 2001, ISSN: 0120047X, 27114732. DOI: [10.38141/10781/023](https://doi.org/10.38141/10781/023) visitado 2 de oct. de 2025. dirección: <https://publicaciones.cenicafe.org/index.php/bot/issue/view/1063>
- [5] N.-E.-A. Mimma, S. Ahmed, T. Rahman y R. Khan, «Fruits Classification and Detection Application Using Deep Learning,» *Scientific Programming*, vol. 2022, Z. Tang, ed., págs. 1-16, nov. de 2022, ISSN: 1875-919X. DOI: [10.1155/2022/4194874](https://doi.org/10.1155/2022/4194874) dirección: <https://doi.org/10.1155/2022/4194874>
- [6] E. I. Muhlisin, R. R. Nurmalasari, L. Kamelia y R. W. Sururie, «Implementation Of Convolutional Neural Network (CNN) in The Android-Based Application for Detecting Coffee Bean Maturity,» en *2024 10th International Conference on Wireless and Tele-matics (ICWT)*, jul. de 2024, págs. 1-5. DOI: [10.1109/icwt62080.2024.10674676](https://doi.org/10.1109/icwt62080.2024.10674676) dirección: <https://doi.org/10.1109/icwt62080.2024.10674676>
- [7] J. N. Wintgens, ed., *Coffee: growing, processing, sustainable production: a guidebook for growers, processors, traders and researchers*, eng. Weinheim Great Britain: Wiley-VCH, 2008, ISBN: 9783527307319 9783527619627.
- [8] S. M. Marín-López, J. Arcila-Pulgarín, E. C. Montoya-Restrepo y C. E. Oliveros-Tascón, «Cambios Físicos Y Químicos Durante La Maduración del Fruto de Café (Coffea arabica L. var. Colombia),» *Cenifcafé*, vol. 54, 3 2003.

- [9] O. Theobald, *Machine learning for absolute beginners: a plain English introduction*, eng, Second edition. [United States]: The author, 2017, OCLC: 1038072041, ISBN: 9781549617218.
- [10] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, eng, Second edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2023, ISBN: 9781492032649 9781492032618.
- [11] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, «You only look once: Unified, real-time object detection,» en *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December, 2016. DOI: [10.1109/CVPR.2016.91](https://doi.org/10.1109/CVPR.2016.91)
- [12] J. Terven, D.-M. Córdova-Esparza y J.-A. Romero-González, «A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS,» en *Machine Learning and Knowledge Extraction*, vol. 5, n.º 4, págs. 1680-1716, nov. de 2023, ISSN: 2504-4990. DOI: [10.3390/make5040083](https://doi.org/10.3390/make5040083) visitado 2 de oct. de 2025. dirección: <https://www.mdpi.com/2504-4990/5/4/83>
- [13] Ultralytics, *Ultralytics YOLO11*, <https://docs.ultralytics.com/es/models/yolo11/>, Accessed: 2025-10-02, 2025.
- [14] I. A. Ansari y V. Bajaj, eds., *Image Processing with Python: A practical approach*. IOP Publishing, jul. de 2024, ISBN: 9780750359245 9780750359221. DOI: [10.1088/978-0-7503-5924-5](https://doi.org/10.1088/978-0-7503-5924-5) visitado 2 de oct. de 2025. dirección: <https://iopscience.iop.org/book/edit/978-0-7503-5924-5>
- [15] *Welcome to Python.org*, en, sep. de 2025. visitado 2 de oct. de 2025. dirección: <https://www.python.org/>
- [16] K. Moore, *Mastering Flutter*, eng, 1st ed. Delhi: BPB Publications, 2025, ISBN: 9789365899177 9789365896039.
- [17] *Dart overview*, oct. de 2025. visitado 2 de oct. de 2025. dirección: <https://dart.dev/overview.md>

- [18] *Advance AI with Open Source*, en-US, sep. de 2025. visitado 2 de oct. de 2025. dirección: <https://www.anaconda.com/>
- [19] *The Jupyter Notebook — Jupyter Notebook 7.5.0b0 documentation*, sep. de 2025. visitado 2 de oct. de 2025. dirección: <https://jupyter-notebook.readthedocs.io/en/latest/notebook.html>
- [20] *Google Colab*, es, sep. de 2025. visitado 2 de oct. de 2025. dirección: <https://colab.research.google.com/>
- [21] *Setting up Visual Studio Code*, en, jul. de 2025. visitado 2 de oct. de 2025. dirección: <https://code.visualstudio.com/docs/setup/setup-overview>
- [22] *Introducción a Android Studio*, es-419-x-mtfrom-en, sep. de 2025. visitado 2 de oct. de 2025. dirección: <https://developer.android.com/studio/intro?hl=es-419>
- [23] *Flutter - Build apps for any screen*, en, sep. de 2025. visitado 2 de oct. de 2025. dirección: <https://flutter.dev/>
- [24] *What is Docker?* en, sep. de 2024. visitado 3 de oct. de 2025. dirección: <https://docs.docker.com/get-started/docker-overview/>
- [25] W. Venema, *Building serverless applications with Google Cloud Run: a real-world guide to building production-ready services*, eng, First edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2020, ISBN: 9781492057093 9781492057048 9781492057062.
- [26] *Qué es Cloud Run*, es-x-mtfrom-en, sep. de 2025. visitado 3 de oct. de 2025. dirección: <https://cloud.google.com/run/docs/overview/what-is-cloud-run?hl=es>
- [27] *Documentación de Compilación | Firebase Documentation*, es-419-x-mtfrom-en, sep. de 2025. visitado 3 de oct. de 2025. dirección: <https://firebase.google.com/docs/build?hl=es-419>
- [28] M. Kleppmann, *Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems*, eng, First edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2017, ISBN: 9781449373320 9781491903117.

- [29] M. A. Tamayo-Monsalve et al., «Coffee Maturity Classification Using Convolutional Neural Networks and Transfer Learning,» *IEEE Access*, vol. 10, págs. 42 971-42 982, 2022, ISSN: 21693536. DOI: [10.1109/ACCESS.2022.3166515](https://doi.org/10.1109/ACCESS.2022.3166515)
- [30] H. C. Bazame, J. P. Molin, D. Althoff y M. Martello, «Detection of coffee fruits on tree branches using computer vision,» *Scientia Agricola*, vol. 80, 2023, ISSN: 1678-992X. DOI: [10.1590/1678-992x-2022-0064](https://doi.org/10.1590/1678-992x-2022-0064) dirección: <https://doi.org/10.1590/1678-992x-2022-0064>
- [31] M. S. Fuentes, N. A. L. Zelaya y J. L. O. Avila, «Coffee Fruit Recognition Using Artificial Vision and neural NETWORKS,» en *2020 5th International Conference on Control and Robotics Engineering (ICCRE)*, sep. de 2020. DOI: [10.1109/iccre49379.2020.9096441](https://doi.org/10.1109/iccre49379.2020.9096441) dirección: <https://doi.org/10.1109/iccre49379.2020.9096441>
- [32] K. Madhavi, Y. S. Babu, G. Ramesh, D. Dua y V. B. Reddy, «Review on Tomato Ripe Detection and Segmentation Using Deep learning Models for Sustainable Agricultural Development,» *E3S Web of Conferences*, vol. 430, S. S. Kumar, ed., pág. 1058, 2023, ISSN: 2267-1242. DOI: [10.1051/e3sconf/202343001058](https://doi.org/10.1051/e3sconf/202343001058) dirección: <https://doi.org/10.1051/e3sconf/202343001058>
- [33] S. Anita y Albarda, «Classification Cherry's Coffee using k-Nearest Neighbor (KNN) and Artificial Neural Network (ANN),» en *2020 International Conference on Information Technology Systems and Innovation (ICITSI)*, IEEE, oct. de 2020, págs. 117-122, ISBN: 978-1-7281-8194-3. DOI: [10.1109/ICITSI50517.2020.9264927](https://doi.org/10.1109/ICITSI50517.2020.9264927)
- [34] N. Huang et al., «Smart agriculture: realtime classification of green coffee beans by using a convolutional neural network,» *IET Smart Cities*, vol. 2, págs. 167-172, 4 oct. de 2020, ISSN: 2631-7680. DOI: [10.1049/iet-smc.2020.0068](https://doi.org/10.1049/iet-smc.2020.0068) dirección: <https://doi.org/10.1049/iet-smc.2020.0068>
- [35] G. Vilcamiza, N. Trelles, L. Vincés y J. Oviden, «A coffee bean classifier system by roast quality using convolutional neural networks and computer vision implemented in an NVIDIA Jetson Nano,» en *2022 Congreso Internacional de Innovación y Tendencias en Ingeniería, CONIITI 2022 - Conference Proceedings*, Institute of Electrical y Electronics

Engineers Inc., 2022, ISBN: 9781665465250. DOI: [10.1109/CONIITI57704.2022.9953636](https://doi.org/10.1109/CONIITI57704.2022.9953636)

- [36] Y. Pei, M. Lian, Y. Jiang, J. Ye, X. Han e Y. Gu, «Real-time Cherry Color Grading Based on Machine Vision,» en *2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP)*, IEEE, dic. de 2019, págs. 1-6, ISBN: 978-1-7281-2345-5. DOI: [10.1109/ICSIDP47821.2019.9173073](https://doi.org/10.1109/ICSIDP47821.2019.9173073)
- [37] L. S. Santana, G. A. e Silva Ferraz, G. H. R. dos Santos, N. L. Bento y R. de Oliveira Faria, «Identification and Counting of Coffee Trees Based on Convolutional Neural Network Applied to RGB Images Obtained by RPA,» *Sustainability*, vol. 15, pág. 820, 1 ene. de 2023, ISSN: 2071-1050. DOI: [10.3390/su15010820](https://doi.org/10.3390/su15010820) dirección: <https://doi.org/10.3390/su15010820>
- [38] C. A. M. de Abreu Júnior et al., «Estimating Coffee Plant Yield Based on Multispectral Images and Machine Learning Models,» *Agronomy*, vol. 12, pág. 3195, 12 dic. de 2022, ISSN: 2073-4395. DOI: [10.3390/agronomy12123195](https://doi.org/10.3390/agronomy12123195) dirección: <https://doi.org/10.3390/agronomy12123195>
- [39] K. P. Lewis y J. D. Espineli, «Classification and detection of nutritional deficiencies in coffee plants using image processing and convolutional neural network (Cnn),» *International Journal of Scientific and Technology Research*, vol. 9, 4 2020, ISSN: 22778616.
- [40] F. Eron, M. Noman, R. R. de Oliveira y A. Chalfun-Junior, «Computer Vision-Aided Intelligent Monitoring of Coffee: Towards Sustainable Coffee Production,» *Scientia Horticulturae*, vol. 327, 2024, ISSN: 03044238. DOI: [10.1016/j.scienta.2024.112847](https://doi.org/10.1016/j.scienta.2024.112847) dirección: <https://doi.org/10.1016/j.scienta.2024.112847>
- [41] T.-H. Hsu et al., «A 0.8 V Intelligent Vision Sensor With Tiny Convolutional Neural Network and Programmable Weights Using Mixed-Mode Processing-in-Sensor Technique for Image Classification,» *IEEE Journal of Solid-State Circuits*, vol. 58, págs. 3266-3274, 11 sep. de 2023, ISSN: 0018-9200. DOI: [10.1109/jssc.2023.3285734](https://doi.org/10.1109/jssc.2023.3285734) dirección: <https://doi.org/10.1109/jssc.2023.3285734>

- [42] L. Yang et al., «Automatic Detection of Banana Maturity—Application of Image Recognition in Agricultural Production,» *Processes*, vol. 12, pág. 799, 4 abr. de 2024, ISSN: 2227-9717. DOI: [10.3390/pr12040799](https://doi.org/10.3390/pr12040799) dirección: <https://doi.org/10.3390/pr12040799>
- [43] P. Xu, Q. Tan, Y. Zhang, X. Zha, S. Yang y R. Yang, «Research on Maize Seed Classification and Recognition Based on Machine Vision and Deep Learning,» *Agriculture*, vol. 12, pág. 232, 2 feb. de 2022, ISSN: 2077-0472. DOI: [10.3390/agriculture12020232](https://doi.org/10.3390/agriculture12020232) dirección: <https://doi.org/10.3390/agriculture12020232>
- [44] M. A. A. Khan, M. A. Rahman, M. L. Hossain y M. T. Habib, «Machine Vision Based Local Hyacinth Bean Breed Recognition Using Convolutional Neural Network,» en *2024 International Conference on Advances in Computing, Communication, Electrical, and Smart Systems (iCACCESS)*, mar. de 2024, págs. 1-6. DOI: [10.1109/icaccess61735.2024.10499455](https://doi.org/10.1109/icaccess61735.2024.10499455) dirección: <https://doi.org/10.1109/icaccess61735.2024.10499455>
- [45] K. Ko, I. Jang, J. H. Choi, J. H. Lim y D. U. Lee, «Stochastic Decision Fusion of Convolutional Neural Networks for Tomato Ripeness Detection in Agricultural Sorting Systems,» *Sensors*, vol. 21, pág. 917, 3 ene. de 2021, ISSN: 1424-8220. DOI: [10.3390/s21030917](https://doi.org/10.3390/s21030917) dirección: <https://doi.org/10.3390/s21030917>
- [46] J. A. M. Rivera y A. M. Reyes-Duke, «Convolutional Neural Network for the Detection of Cocoa Maturity with an Approach for the Analysis of Images Captured at Different Distances,» en *Engineering Headway*, vol. 12, sep. de 2024, págs. 3-8. DOI: [10.4028/p-jm3tgd](https://doi.org/10.4028/p-jm3tgd) dirección: <https://doi.org/10.4028/p-jm3tgd>
- [47] H. Liu, H. Sun, M. Li y M. Iida, «Application of Color Featuring and Deep Learning in Maize Plant Detection,» *Remote Sensing*, vol. 12, pág. 2229, 14 jul. de 2020, ISSN: 2072-4292. DOI: [10.3390/rs12142229](https://doi.org/10.3390/rs12142229) dirección: <https://doi.org/10.3390/rs12142229>
- [48] P. Lottes, J. Behley, A. Milioto y C. Stachniss, «Fully Convolutional Networks With Sequential Information for Robust Crop and Weed Detection in Precision Farming,» *IEEE Robotics and Automation Letters*, vol. 3, págs. 2870-2877, 4 sep. de 2018, ISSN:

2377-3766. DOI: [10.1109/lra.2018.2846289](https://doi.org/10.1109/lra.2018.2846289) dirección: <https://doi.org/10.1109/lra.2018.2846289>

- [49] M. N. Ratnayake, A. G. Dyer y A. Dorin, «Towards Computer Vision and Deep Learning Facilitated Pollination Monitoring for Agriculture,» en *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, sep. de 2021, págs. 2915-2924. DOI: [10.1109/cvprw53098.2021.00327](https://doi.org/10.1109/cvprw53098.2021.00327) dirección: <https://doi.org/10.1109/cvprw53098.2021.00327>
- [50] B. Darwin, P. Dharmaraj, S. Prince, D. E. Popescu y D. J. Hemanth, «Recognition of Bloom/Yield in Crop Images Using Deep Learning Models for Smart Agriculture: A Review,» *Agronomy*, vol. 11, pág. 646, 4 mar. de 2021, ISSN: 2073-4395. DOI: [10.3390/agronomy11040646](https://doi.org/10.3390/agronomy11040646) dirección: <https://doi.org/10.3390/agronomy11040646>
- [51] S. Jayaweera et al., «MangoDB - A TJC Mango Dataset for Deep-Learning-Based on Classification and Detection in Precision Agriculture,» en *2024 4th International Conference on Advanced Research in Computing (ICARC)*, feb. de 2024, págs. 115-120. DOI: [10.1109/icarc61713.2024.10499698](https://doi.org/10.1109/icarc61713.2024.10499698) dirección: <https://doi.org/10.1109/icarc61713.2024.10499698>
- [52] A. Ahmed y A. A. Rafique, «Deep Network for Smart Precision Agriculture through Segmentation and classification of Crops,» en *2022 19th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, ago. de 2022, págs. 502-507. DOI: [10.1109/ibcast54850.2022.9990351](https://doi.org/10.1109/ibcast54850.2022.9990351) dirección: <https://doi.org/10.1109/ibcast54850.2022.9990351>
- [53] D. I. Patrício y R. Rieder, «Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review,» *Computers and Electronics in Agriculture*, vol. 153, págs. 69-81, sep. de 2018, ISSN: 0168-1699. DOI: [10.1016/j.compag.2018.08.001](https://doi.org/10.1016/j.compag.2018.08.001) dirección: <https://doi.org/10.1016/j.compag.2018.08.001>
- [54] J. A. Fracarolli, F. F. A. Pavarin, W. Castro y J. Blasco, «Computer vision applied to food and agricultural products,» *REVISTA CIÊNCIA AGRONÔMICA*, vol. 51, 5 2020, ISSN: 1806-6690. DOI: [10.5935/1806-6690.20200087](https://doi.org/10.5935/1806-6690.20200087) dirección: <https://doi.org/10.5935/1806-6690.20200087>

- [55] M. Pyingkodi, K. Thenmozhi, M. Karthikeyan, K. Chitra, N. R. W. Blessing y S. Kumar, «Fruits Quality Detection using Deep Learning Models: A Meta- Analysis,» en *2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC)*, ago. de 2022, págs. 1-8. DOI: [10 . 1109 / icesc54411 . 2022 . 9885289](https://doi.org/10.1109/icesc54411.2022.9885289) dirección: <https://doi.org/10.1109/icesc54411.2022.9885289>
- [56] N. Ohura et al., «Convolutional neural networks for wound detection: the role of artificial intelligence in wound care,» *Journal of Wound Care*, vol. 28, S13-S24, Sup10 oct. de 2019, ISSN: 0969-0700. DOI: [10 . 12968 / jowc . 2019 . 28 . sup10 . s13](https://doi.org/10.12968/jowc.2019.28.sup10.s13) dirección: <https://doi.org/10.12968/jowc.2019.28.sup10.s13>
- [57] A. Tsuboi et al., «Artificial intelligence using a convolutional neural network for automatic detection of smallbowel angioectasia in capsule endoscopy images,» *Digestive Endoscopy*, vol. 32, págs. 382-390, 3 oct. de 2019, ISSN: 0915-5635. DOI: [10.1111/den.13507](https://doi.org/10.1111/den.13507) dirección: <https://doi.org/10.1111/den.13507>
- [58] S. R. Ziyad, M. Alharbi y M. Altulyan, «Artificial Intelligence Model for Alzheimers Disease Detection with Convolution Neural Network for Magnetic Resonance Images,» *Journal of Alzheimer's Disease*, vol. 93, págs. 235-245, 1 mayo de 2023, ISSN: 1387-2877. DOI: [10 . 3233 / jad - 221250](https://doi.org/10.3233/jad-221250) dirección: <https://doi.org/10.3233/jad-221250>
- [59] Ultralytics, *Comprensión de los pasos clave en un proyecto de visión artificial*, <https://docs.ultralytics.com/es/guides/steps-of-a-cv-project/>, Accessed: 2025-10-02, 2025.
- [60] D. E. Manrique Santos, «Sistema embebido para la identificación de defectos en granos de café verde, mediante procesamiento de imágenes y algoritmos de inteligencia artificial,» Tesis de Maestría, Tecnológico Nacional de México Campus Orizaba, Orizaba, México, 2023.
- [61] L. A. Alvarado Escua, «Sistema de visión artificial para determinar la densidad de pollos de engorde,» Tesis de Maestría, Tecnológico Nacional de México Campus Orizaba, Orizaba, México, 2023.
- [62] Roboflow, *Roboflow*, <https://roboflow.com/>, Accessed: 2025-10-02, 2025.

- [63] Kaggle, *Coffee Fruit Maturity*, <https://www.kaggle.com/datasets/cienciacafero/coffee-fruit-maturity>, Accessed: 2025-10-02, 2023.
- [64] Roboflow, *Coffee Cherry Beans Detection Computer Vision Model*, <https://universe.roboflow.com/chess-mlx7u/coffee-cherry-beans-detection>, Accessed: 2025-10-02, 2024.
- [65] Roboflow, *Coffee Dataset 2 Computer Vision Dataset*, <https://universe.roboflow.com/alfito-dwi-putra/coffee-dataset-2>, Accessed: 2025-10-02, 2024.
- [66] Roboflow, *coffee fruit Computer Vision Dataset*, <https://universe.roboflow.com/seongchan-park/coffee-fruit>, Accessed: 2025-10-02, 2024.
- [67] Ultralytics, *Guía de recolección y anotación de datos: Evitando sesgos en la recolección de datos*, <https://docs.ultralytics.com/es/guides/data-collection-and-annotation/#avoiding-bias-in-data-collection>, Accessed: 2025-10-02, 2025.
- [68] Roboflow, *Create a Dataset Version*, <https://docs.roboflow.com/datasets/dataset-versions/create-a-dataset-version#readjusting-train-validation-test-splits>, Accessed: 2025-10-02, 2025.
- [69] Roboflow, *Train, Validation, Test Split for Machine Learning*, <https://blog.roboflow.com/train-test-split/>, Accessed: 2025-10-02, 2025.
- [70] I. Goodfellow, A. Courville e Y. Bengio, *Deep learning* (Adaptive computation and machine learning), eng. Cambridge, Massachusetts: The MIT Press, 2016, ISBN: 9780262337373.
- [71] Roboflow, *What is Data Augmentation? The Ultimate Guide*, <https://blog.roboflow.com/data-augmentation/>, Accessed: 2025-10-02, 2023.
- [72] F. Chollet, *Deep Learning with Python*. Shelter Island, NY: Manning, 2018.
- [73] Ultralytics, *Model Training with Ultralytics YOLO*, <https://docs.ultralytics.com/modes/train/#train-settings>, Accessed: 2025-10-02, 2025.
- [74] Ultralytics, *Model Validation with Ultralytics YOLO*, <https://docs.ultralytics.com/modes/val/>, Accessed: 2025-10-02, 2025.

- [75] T.-Y. Lin et al., «Microsoft COCO: Common objects in context,» en *Computer vision – ECCV 2014*, Springer, 2014, págs. 740-755. DOI: [10.1007/978-3-319-10602-1_48](https://doi.org/10.1007/978-3-319-10602-1_48)
- [76] Ultralytics, *Insights on Model Evaluation and Fine-Tuning*, <https://docs.ultralytics.com/guides/model-evaluation-insights/>, Accessed: 2025-10-02, 2025.
- [77] D. Olson y D. Delen, *Advanced Data Mining Techniques*. Lincoln, NE: Springer, 2008.
- [78] Ultralytics, *Performance Metrics Deep Dive*, <https://docs.ultralytics.com/guides/yolo-performance-metrics/>, Accessed: 2025-10-02, 2025.
- [79] Ultralytics, *A Guide on Model Testing*, <https://docs.ultralytics.com/guides/model-testing/>, Accessed: 2025-10-02, 2025.
- [80] Ultralytics, *Model Prediction with Ultralytics YOLO*, <https://docs.ultralytics.com/modes/predict/>, Accessed: 2025-10-02, 2025.
- [81] Ultralytics, *Exportación de modelos con Ultralytics YOLO*, es, oct. de 2025. visitado 2 de oct. de 2025. dirección: <https://docs.ultralytics.com/es/modes/export/>
- [82] Ultralytics, *Exportación ONNX para modelos YOLO11*, es, oct. de 2025. visitado 2 de oct. de 2025. dirección: <https://docs.ultralytics.com/es/integrations/onnx/>
- [83] Ultralytics, *Implemente un modelo YOLO pre-entrenado con Ultralytics en Vertex AI para la inferencia*, <https://docs.ultralytics.com/es/guides/vertex-ai-deployment-with-docker/>, Accessed: 2025-10-02, 2025.
- [84] Google Cloud. «Almacenar imágenes de contenedor Docker en Artifact Registry.» Documentación oficial de Google Cloud sobre Artifact Registry para Docker, visitado 20 de ene. de 2024. dirección: <https://cloud.google.com/artifact-registry/docs/docker/store-docker-container-images?hl=es>
- [85] Google Cloud. «Envía y extrae imágenes.» Documentación oficial de Google Cloud sobre Artifact Registry para enviar y extraer imágenes de contenedor con Docker. dirección: <https://cloud.google.com/artifact-registry/docs/docker/pushing-and-pulling?hl=es-419>

- [86] *Agregar Firebase a tu app de Flutter*, es-419-x-mtfrom-en, sep. de 2025. visitado 3 de oct. de 2025. dirección: <https://firebase.google.com/docs/flutter/setup?hl=es-419&platform=android>
- [87] *Primeros pasos: Escribe, prueba e implementa tus primeras funciones*, es-419-x-mtfrom-en, sep. de 2025. visitado 3 de oct. de 2025. dirección: <https://firebase.google.com/docs/functions/get-started?hl=es-419>
- [88] *<uses-sdk>*, es-419-x-mtfrom-en, sep. de 2025. visitado 2 de oct. de 2025. dirección: <https://developer.android.com/ndk/guides?hl=es-419>
- [89] *Cómo comenzar a usar el NDK*, es-419-x-mtfrom-en, sep. de 2025. visitado 2 de oct. de 2025. dirección: <https://developer.android.com/guide/topics/manifest/uses-sdk-element?hl=es-419#min>
- [90] *Build and release an Android app*, en, sep. de 2025. visitado 2 de oct. de 2025. dirección: <https://docs.flutter.dev/deployment/android>

PRODUCTOS ACADÉMICOS

Durante la realización de la presente investigación, se generaron varios productos académicos que contribuyen al conocimiento de las área de vision computacional y electrónica, las cuales son:

- Se generó un conjunto de datos de las cerezas de café para el entrenamiento de redes neuronales convolucionales enfocadas a la vision computacional.
- Se generó un modelo de red neuronal convolucional para la detección y clasificación de las cerezas de café de acuerdo a su maduración.
- Se elaboró el despliegue en la nube del modelo de red neuronal convolucional usando Docker, Google Cloud y Firebase.
- Se elaboró una aplicación móvil para sistemas Android, la cual muestra la inferencia del modelo de red neuronal convolucional.
- Se realizó un artículo científico titulado ‘DeepVision for Precision Farming: Automated Ripeness Grading of Coffee Cherries Using Deep Learning Algorithms’ en la revista ‘Agroproductividad’ y que está en proceso de publicación al momento de realizar la presente investigación, donde se explica el desarrollo del entrenamiento de la red neuronal convolucional utilizada para el sistema clasificador de madurez de cerezas de café de la actual investigación.
- Al momento de la realización de la investigación, está en procesamiento el registro de la aplicación móvil y el modelo de red neuronal convolucional en la categoría de programa de computo y el conjunto de datos en la categoría de base de datos ante el Instituto Nacional del Derecho de Autor (INDAUTOR).

ANEXOS

4.3 Anexo A : Código de la aplicación móvil

4.3.1 main.dart

Listing 4.3: Fragmento del código main.dart

```
1 Future<void> main() async {
2   WidgetsFlutterBinding.ensureInitialized();
3   await Firebase.initializeApp(options:
4     DefaultFirebaseOptions.currentPlatform);
5
6   runApp(MyApp(photoLocations: []));
7 }
8 class MyApp extends StatelessWidget {
9   final List<LatLng> photoLocations;
10
11   const MyApp({
12     super.key,
13     required this.photoLocations,
14   });
15
16   GoRouter _router(List<LatLng> photoLocations) {
17     return GoRouter(
18       initialLocation: '/login_page',
19       routes: [
20         GoRoute(
21           path: '/login_page',
22           builder: (context, state) => LoginPage(),
23         ),
24         GoRoute(
25           path: '/register_page',
26           builder: (context, state) => RegistroPage(),
```

```

26     ),
27     GoRoute(
28       path: '/inicio',
29       builder: (context, state) => Pantalla_00(
30         photoLocations: photoLocations
31       ),
32     ),
33     GoRoute(
34       path: '/galeria',
35       builder: (context, state) => Pantalla_02(),
36     ),
37     GoRoute(
38       path: '/configuracion',
39       builder: (context, state) => Pantalla_03(),
40     ),
41     GoRoute(
42       path: '/dashboard',
43       builder: (context, state) => InfoPage(),
44     ),
45   ],
46 );
47 }
48 }

```

4.3.2 navigation_bar_widget.dart

Listing 4.4: Fragmento del código navigation_bar_widget.dart

```

1 @override
2 Widget build(BuildContext context) {
3   return Scaffold(
4     bottomNavigationBar: NavigationBar(
5       onDestinationSelected: (int index) {

```

```

6      setState(() {
7          currentPageIndex = index;
8      });
9      _saveLastPageIndex(index); // Guardar el indice
      seleccionado
10  },
11  selectedIndex: currentPageIndex,
12  destinations: const <Widget>[
13      NavigationDestination(
14          icon: Icon(Icons.home),
15          label: 'Inicio',
16      ),
17      NavigationDestination(
18          icon: Icon(Icons.map),
19          label: 'Mapa',
20      ),
21      NavigationDestination(
22          icon: Icon(Icons.image_rounded),
23          label: 'Galeria',
24      ),
25      NavigationDestination(
26          icon: Icon(Icons.settings),
27          label: 'Configuracion',
28      ),
29  ],
30  ),
31  body: SafeArea(
32      child: IndexedStack(
33          index: currentPageIndex,
34          children: <Widget>[
35              Pantalla_00(photoLocations: widget.photoLocations),
36              Pantalla_01(userId: _currentUserId),

```

```

37         Pantalla_02(),
38         Pantalla_03(),
39     ],
40 ),
41 ),
42 );
43 }

```

4.3.3 login_page.dart

Listing 4.5: Fragmento del código login_page.dart

```

1 FilledButton googleButton(BuildContext context) {
2   return FilledButton.tonalIcon(
3     label: const Text('Iniciar con Google'),
4     onPressed: () async {
5       // Mostrar indicador de carga
6       showDialog(
7         context: context,
8         barrierDismissible: false,
9         builder: (context) => const Center(
10          child: Card(
11            child: Padding(
12              padding: EdgeInsets.all(20),
13              child: Column(
14                mainAxisAlignment: MainAxisAlignment.min,
15                children: [
16                  CircularProgressIndicator(),
17                  Text('Iniciando sesion con Google...'),
18                ],
19              ),
20            ),
21          ),

```



```

22     ),
23   );
24   try {
25     final user = await AuthService().loginGoogle();
26   } on FirebaseAuthException catch (error) {
27     print('FirebaseAuthException: ${error.code} -
28         ${error.message}');
29   } catch (error) {
30     print('Error: $error');
31   }
32 },
33 );
34 }

```

4.3.4 register_page.dart

Listing 4.6: Fragmento del código register_page.dart

```

1 ElevatedButton(
2   onPressed: () async {
3     if (_formKey.currentState?.saveAndValidate() ?? false) {
4       final v = _formKey.currentState?.value;
5
6       try {
7         // Crear cuenta en Firebase Auth
8         final uid = await _auth.createAccount(
9           v?['user'], v?['pass']);
10
11        // Guardar datos del usuario en Firestore
12        await FirebaseFirestore.instance
13          .collection('user')
14          .doc(uid)
15          .set({

```

```

16         'email': v?['user'],
17         'uid': uid,
18     });
19
20     context.pop(); // Cerrar el indicador
21
22     if (mounted) {
23         context.go('/navigation');
24     }
25     } catch (error) {
26         context.pop();
27     }
28 }
29 },
30 child: const Text(
31     'CREAR CUENTA',
32     style: TextStyle(color: Colors.white),
33 ),
34 ),

```

4.3.5 pref_usuarios.dart

Listing 4.7: Fragmento del código pref_usuarios.dart

```

1 class PreferenciasUsuario {
2     /// Instancia unica de SharedPreferences
3     static late SharedPreferences _prefs;
4
5     /// Claves para las preferencias
6     static const String _keyUltimaPagina = 'ultimaPagina';
7
8     /// Inicializa las preferencias
9     static Future<void> init() async {

```

```

10     try {
11         _prefs = await SharedPreferences.getInstance();
12     } catch (e) {
13         print('Error al inicializar las preferencias: $e');
14         rethrow; // Lanza el error para que el flujo superior lo
15                 // maneje
16     }
17 }
18 /// Getter y Setter para la ultima pagina visitada
19 String get ultimaPagina {
20     return _prefs.getString(_keyUltimaPagina) ?? 'Login';
21 }
22 set ultimaPagina(String value) {
23     _prefs.setString(_keyUltimaPagina, value);
24 }

```

4.3.6 auth.dart

Listing 4.8: Fragment del código auth.dart

```

1 class AuthService {
2     final FirebaseAuth _auth = FirebaseAuth.instance;
3     final GoogleSignIn _googleSignIn = GoogleSignIn();
4     final FirebaseFirestore _firestore =
5         FirebaseFirestore.instance;
6
7     /// Inicia sesion con Google.
8     /// Retorna el usuario de Firebase o lanza un mensaje de error.
9     Future<User?> loginGoogle() async {
10         try {
11             // Iniciar el flujo de autenticacion de Google

```

```

11     final GoogleSignInAccount? googleUser = await
        _googleSignIn.signIn();
12     if (googleUser == null) {
13         return null;
14     }
15     // Obtener los detalles de autenticacion
16     final GoogleSignInAuthentication googleAuth =
17     await googleUser.authentication;
18     // Crear credencial de Firebase
19     final AuthCredential credential =
        GoogleAuthProvider.credential(
20         accessToken: googleAuth.accessToken, idToken:
            googleAuth.idToken,
21         );
22     // Iniciar sesion en Firebase con la credencial
23     final UserCredential userCredential = await
        _auth.signInWithCredential(credential);
24     final User? user = userCredential.user;
25
26     return user;
27 } on FirebaseAuthException catch (e) {
28     throw _handleFirebaseAuthError(e.code);
29 } catch (e) {
30     throw 'Error al iniciar sesion con Google:
        ${e.toString()}';
31 }
32 }
33 }

```

4.3.7 storage_services.dart

Listing 4.9: Fragmento del código storage_services.dart

```

1 final FirebaseStorage storage = FirebaseStorage.instance;
2 final FirebaseAuth auth = FirebaseAuth.instance;
3
4 /// Retorna la URL de la imagen procesada o null si no existe
5 Future<String?> obtenerImagenProcesada(String photoId) async {
6     final user = auth.currentUser;
7     if (user == null) {
8         throw Exception('Usuario no autenticado');
9     }
10
11     try {
12         final String userId = user.uid;
13         final Reference resultFolder =
14             storage.ref().child('resultados/$userId/photos/$photoId');
15
16         // Listar archivos en la carpeta de resultados
17         final ListResult files = await resultFolder.listAll();
18
19         // Buscar el primer archivo de imagen
20         for (var item in files.items) {
21             if (item.name.toLowerCase().endsWith('.jpg') ||
22                 item.name.toLowerCase().endsWith('.jpeg') ||
23                 item.name.toLowerCase().endsWith('.png')) {
24                 String url = await item.getDownloadURL();
25                 print("Imagen procesada encontrada: $url");
26                 return url;
27             }
28         }
29         print("No se encontro imagen procesada para photoId:
30             $photoId");
31         return null;

```

```

30   } catch (e) {
31     print("Error al obtener imagen procesada: $e");
32     return null;
33   }
34 }

```

4.3.8 Pantalla_00

Listing 4.10: Fragmento del código pantalla_00.dart

```

1  class _Pantalla_00State extends State<Pantalla_00> {
2
3    @override
4    void initState() {
5      super.initState();
6      _initializeMarkers();
7    }
8
9    /// Agrega una nueva ubicacion
10   Future<void> _addLocation() async {
11     setState(()=>_isLoading = true);
12     try {
13       // Obtener la ubicacion actual
14       LatLng currentLocation = await _getCurrentLocation();
15       String name = _generatePlaceName();
16       // Crear el objeto Place
17       final draft = PlaceDraft(
18         name: name,
19         lat: currentLocation.latitude,
20         lon: currentLocation.longitude,
21       );
22       // Navegar a la pantalla de camara
23       if (!mounted) return;

```

```

24     context.push('/camera');
25   } catch (e) {
26     _showErrorSnackBar('Error al agregar ubicacion: $e');
27   } finally {
28     if (mounted) setState(()=>_isLoading = false);
29   }
30 }
31 }

```

4.3.9 camera_screen.dart

Listing 4.11: Fragmento del código camera_screen.dart

```

1  class _CameraScreenState extends State<CameraScreen> {
2    final ImagePicker _picker = ImagePicker();
3
4    Future<void> _openCamera() async {
5      try {
6        final status = await Permission.camera.request();
7        if (status.isDenied || status.isPermanentlyDenied) {
8          if (mounted) context.go('/navigation');
9          return;
10       }
11       final picture = await _picker.pickImage(source:
12         ImageSource.camera);
13       if (picture != null && picture.path.isNotEmpty) {
14         final path = picture.path;
15         // Navegar a la pantalla de visualizacion de imagen si
16         se toma la foto
17         if (!mounted) return;
18         context.go('/picture');
19       } else {
20         if (!mounted) return;

```

```

19     context.go('/navigation'); // Regresa si se cancela
20   }
21   } catch (e) {
22     _showSnackBar('Error al abrir la camara: $e');
23     if (mounted) context.pop('/navigation'); // Regresa si hay
        error
24   }
25 }
26 }

```

4.3.10 picture_screen.dart

Listing 4.12: Fragmento de código picture_screen.dart

```

1 class _PictureScreenState extends State<PictureScreen> {
2   bool _isSaving = false; // Estado para indicar si se esta
        guardando
3   String? _imageUrl; // URL de la imagen subida
4   String? _error; // Mensaje de error si ocurre un problema
        durante la subida
5
6   /// Subir la imagen a Firebase Storage y guardar la URL
7   Future<void> _uploadAndSave() async {
8     setState(() {
9       _isSaving = true;
10      _error = null;
11    });
12    try {
13      final user = FirebaseAuth.instance.currentUser;
14
15      // Guardar documento en Firestore
16      final docRef = await FirebaseFirestore.instance
17        .collection('user')

```



```

18         .doc(user.uid)
19         .collection('photos')
20         .add({
21             'name': widget.draft.name,
22             'lat': widget.draft.lat,
23             'lon': widget.draft.lon,
24             'userId': user.uid,
25             'createdAt': FieldValue.serverTimestamp(),
26         });
27         final photoId = docRef.id;
28
29         // Subir imagen con el photoId.
30         final url = await subirImagen(
31             File(widget.imagePath),
32             photoId
33         );
34         _imageUrl=url;
35
36         if (!mounted) return;
37         ScaffoldMessenger.of(context).showSnackBar(
38             const SnackBar(
39                 content: Text('Imagen guardada correctamente.'),
40                 duration: Duration(seconds: 2),
41             ),
42         );
43         // Regresar a navegacion
44         context.go('/navigation');
45     } catch (e) {
46         setState(() => _error = 'Error al subir la imagen: $e');
47     } finally{
48         if (mounted) setState(() => _isSaving = false);
49     }

```

```
50 }  
51 }
```

4.3.11 Pantalla_01

Listing 4.13: Fragmento de código pantalla_01.dart

```
1 class Pantalla_01 extends StatelessWidget {  
2   final String userId; // userId del usuario actual  
3  
4   const Pantalla_01({super.key, required this.userId});  
5  
6   @override  
7   Widget build(BuildContext context) {  
8     return SafeArea(  
9       child: Scaffold(  
10        appBar: AppBar(  
11          title: const Text('Mapa de Detecciones'),  
12          centerTitle: true,  
13        ),  
14        body: MainMap(), // Pasamos el userId al MainMap  
15      ),  
16    );  
17  }  
18 }
```

4.3.12 mapa.dart

Listing 4.14: Fragmento del código mapa.dart

```
1 class _MainMapState extends State<MainMap> {  
2   bool _isLoading = true; // Estado para indicar si el mapa se  
   esta inicializando
```

```

3 String? _errorMessage; // Mensaje de error si ocurre un
   problema
4
5 /// Inicializa el mapa obteniendo la ubicacion actual y datos
   de la base de datos
6 Future<void> _initializeMap() async {
7     try {
8         // (1) Obtener la ubicacion actual (marcador azul)
9         Position currentPosition = await _getCurrentLocation();
10        // Agregar marcador para la ubicacion actual
11        _addMarker(
12            LatLng(currentPosition.latitude,
13                    currentPosition.longitude),
14            'currpos',
15            'Estas aqui',
16            BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueAzure),
17        );
18    } catch (e) {
19        setState(() {
20            _errorMessage = 'Error: $e'; // Mostrar el error si
21            ocurre uno
22            _isLoading = false;
23        });
24    }
25 }
26
27 /// Construccion de la UI del mapa
28 @override
29 Widget build(BuildContext context) {
30     return Scaffold(
31         body: GoogleMap(
32             mapType: MapType.normal,

```

```

31     ),
32   );
33 }
34 }

```

4.3.13 Pantalla_02

Listing 4.15: Fragmento del código pantalla_02.dart

```

1  class Pantalla_02 extends StatelessWidget {
2    const Pantalla_02({super.key});
3
4    @override
5    Widget build(BuildContext context) {
6      return Scaffold(
7        appBar: AppBar(
8          title: const Text('Galeria de Detecciones'),
9          centerTitle: true,
10       ),
11       body: const SafeArea(
12         child: ManagePlace(),
13       ),
14     );
15   }
16 }

```

4.3.14 manage_place.dart

Listing 4.16: Fragmento del código manage_place.dart

```

1  class PlacesList extends StatelessWidget {
2    const PlacesList({super.key});
3

```

```

4  @override
5  Widget build(BuildContext context) {
6      return StreamBuilder<QuerySnapshot<Place>>(
7          stream: query.snapshots(),
8          builder: (context, snap) {
9              return GridView.builder(
10                 gridDelegate: const
11                     SliverGridDelegateWithFixedCrossAxisCount(
12                         crossAxisCount: 2,
13                         mainAxisSpacing: 8,
14                         crossAxisSpacing: 8,
15                         childAspectRatio: 0.78,
16                     ),
17                 itemCount: docs.length,
18                 itemBuilder: (context, i) {
19                     final place = docs[i].data();
20
21                     return Card(
22                         clipBehavior: Clip.hardEdge,
23                         elevation: 2,
24                         child: InkWell(
25                             onTap: () {
26                                 showDialog(
27                                     context: context,
28                                     builder: (_) => PlaceDialog().buildAlert(),
29                                 );
30                             },
31                         ),
32                     );
33                 },
34             );

```

```
35     );  
36   }  
37 }
```

4.3.15 place.dart

Listing 4.17: Fragmento del código place.dart

```
1  class Place {  
2    final String id;  
3    final String name;  
4    final double lat;  
5    final double lon;  
6    final String imageUrl;  
7    final String userId;  
8    final DateTime createdAt;  
9  
10   Place({  
11     required this.id,  
12     required this.name,  
13     required this.lat,  
14     required this.lon,  
15     required this.imageUrl,  
16     required this.userId,  
17     required this.createdAt,  
18   });  
19  
20   /// Convierte el objeto a un Map para guardarlo en la base de  
21   /// datos (excluye el 'id' para inserciones).  
22   Map<String, dynamic> toMapForCreate() {  
23     return {  
24       'name': name,  
25       'lat': lat,  
26     };  
27   }  
28 }
```

```

25     'lon': lon,
26     'imageUrl': imageUrl,
27     'userId': userId,
28     'createdAt': FieldValue.serverTimestamp(),
29   };
30 }
31 }

```

4.3.16 place_draft.dart

Listing 4.18: Fragmento del código place_draft.dart

```

1  class PlaceDraft {
2    final String name;
3    final double lat;
4    final double lon;
5
6    const PlaceDraft({
7      required this.name,
8      required this.lat,
9      required this.lon,
10   });
11 }

```

4.3.17 place_dialog.dart

Listing 4.19: Fragmento del código place_Dialog.dart

```

1  class PlaceDialog {
2    final Place place; // Objeto 'Place' que contiene los datos
                        // del lugar (nombre, latitud, longitud e imagen).
3    PlaceDialog(this.place);
4

```

```

5  Widget buildAlert(BuildContext context) {
6      return AlertDialog(
7          child: Column(
8              children: [
9                  // Campo de texto para mostrar el nombre del lugar.
10                 SelectableText("Nombre: ${place.name}"),
11                 const SizedBox(height: 6),
12                 SelectableText("Latitud: ${place.lat}"),
13                 const SizedBox(height: 6),
14                 SelectableText("Longitud: ${place.lon}"),
15                 const SizedBox(height: 10),
16             ],
17         ),
18         // Acciones del cuadro de dialogo.
19         actions: [
20             // Boton para cerrar el cuadro de dialogo.
21             TextButton(
22                 onPressed: () => context.pop(), // Cierra el cuadro de
23                 dialogo al presionar el boton.
24                 child: const Text('Cerrar'), // Etiqueta del boton.
25             ),
26         ],
27     );
28 }

```

4.3.18 Pantalla_03

Listing 4.20: Fragmento del código pantalla_03.dart

```

1  class _Pantalla_03State extends State<Pantalla_03> {
2      @override
3      Widget build(BuildContext context) {

```



```

4      final currentUser = FirebaseAuth.instance.currentUser;
5
6      return Scaffold(
7        body: Column(
8          children: <Widget>[
9            // Card de informacion
10           Card(
11             child: ListTile(
12               title: const Text('Acerca de'),
13               onTap: () {
14                 showDialog(
15                   context: context,
16                   builder: (context) => const InfoPage(),
17                 );
18               },
19             ),
20           ),
21           // Card de cerrar sesion
22           Card(
23             child: ListTile(
24               title: const Text('Cerrar sesion'),
25               onTap: () async => _showLogoutDialog(),
26             ),
27           ),
28         ],
29       ),
30     );
31   }
32 }

```

4.3.19 informacion.dart

Listing 4.21: Fragmento del código informacion.dart

```
1 class _InfoPageState extends State<InfoPage> {
2   @override
3   Widget build(BuildContext context) {
4     return AlertDialog(
5       // Titulo del cuadro de dialogo.
6       title: Text('Acerca de'),
7       // Contenido del cuadro de dialogo.
8       content: SingleChildScrollView(
9         child: Column(
10          children: [
11            Text(
12              'Aplicacion movil para la deteccion de la madurez
13                de cerezas de cafe en imagenes capturadas por
14                el dispositivo mediante inteligencia
15                artificial.',
16              textAlign: TextAlign.justify,
17            ),
18            Center(
19              child: Text(
20                'Maestria en Ingenieria Electronica\nTecnologico
21                Nacional de Mexico\nCampus Orizaba\n2025',
22                textAlign: TextAlign.center,
23              ),
24            ),
25          ],
26        ),
27      ),
28      // Acciones del cuadro de dialogo.
29      actions: <Widget>[
30        TextButton(
```

```

27         onPressed: () => context.pop(), // Cierra el cuadro de
           dialogo al presionar el boton.
28         child: const Text('Cerrar'), // Etiqueta del boton.
29     ),
30 ],
31 );
32 }
33 }

```

4.4 Anexo B : Código de la API REST

4.4.1 main.py

Listing 4.22: Código main.py

```

1 # FastAPI
2 import sys
3 import os
4 from fastapi import FastAPI, HTTPException, status, Response,
   UploadFile, File
5 from pydantic import BaseModel
6 from typing import Any, Dict, Optional, List
7 from app import is_model_ready, run_inference,
   get_image_from_bytes, get_bytes_from_image,
   get_annotated_image
8 import uvicorn
9 import base64
10 from loguru import logger
11
12 # Configurar logger
13 # Eliminar logger por defecto
14 logger.remove()

```

```

15 # Configurar el logger para output en consola (necesario para
    Cloud Run)
16 # - colorize: colorea la salida
17 # - format: formato personalizado del log
18 # - level: nivel minimo de log (INFO)
19 # - serialize: no serializar a JSON
20 logger.add(
21     sys.stdout,
22     colorize=True,
23     format="<green>{time:HH:mm:ss}</green> |
        <level>{message}</level>",
24     level="INFO",
25     serialize=False,
26 )
27 # Agrega logger a archivo local para debugging
28 # - rotation: rota el archivo cuando alcanza 1MB
29 # - level: DEBUG (mas detallado que INFO)
30 # - compression: comprime los logs rotados en ZIP
31 logger.add("log.log", rotation="1 MB", level="DEBUG",
    compression="zip")
32
33 # Crea la instancia principal de FastAPI con metadatos:
34 # - titulo, version y descripcion de la API
35 # - URLs para documentacion automatica
36 app = FastAPI(
37     title="YOL011 Inference Server",
38     version="1.0.0",
39     description="**Deteccion de cereza de cafe usando YOL011**",
40     docs_url="/docs",
41     redoc_url="/redoc",
42     openapi_url="/openapi.json"
43 )

```

```

44
45 # Lee variables de entorno para configuracion en Cloud Run
46 # Valores por defecto si no estan definidas las variables
47 PORT = int(os.getenv("PORT", "8000"))
48 AIP_HEALTH_ROUTE = os.getenv("AIP_HEALTH_ROUTE", "/health")
49 AIP_PREDICT_ROUTE = os.getenv("AIP_PREDICT_ROUTE", "/predict")
50
51 # Define limite maximo de tamano para requests (10MB)
52 MAX_REQUEST_SIZE = 10 * 1024 * 1024 # 10 MB in bytes
53
54 # Modelos pydantic para request/response
55 class Detection(BaseModel): #Define una deteccion individual
56     class_name: str # Ej: "cereza_madura"
57     confidence: float # Ej: 0.95 (95% de confianza)
58     bbox: Dict[str, float] # Ej: {"xmin": 10, "ymin": 20,
59         "xmax": 50, "ymax": 60}
60
61 class PredictionResult(BaseModel): #Define el resultado
62     #completo de una prediccion
63     detections: List[Detection] # Lista de detecciones
64     detection_count: int # Cantidad total de
65         detecciones
66     annotated_image: Optional[str] = None # Imagen anotada en
67         base64 (opcional)
68
69 class PredictionRequest(BaseModel): #Lo que el
70     #cliente envia
71     instances: List[Dict[str, Any]] # Lista de
72         imagenes/instancias
73     parameters: Optional[Dict[str, Any]] = None # Parametros
74         opcionales
75
76

```

```

69 class PredictionResponse(BaseModel):      #Lo que la API devuelve
70     predictions: List[PredictionResult]    # Lista de resultados
71                                           por cada instancia
72
73 # Endpoint de health check que:
74 # - Verifica si el modelo esta listo
75 # - Retorna 503 si no esta listo
76 # - Retorna 200 con status "healthy" si esta listo
77 @app.get(AIP_HEALTH_ROUTE, status_code=status.HTTP_200_OK)
78 def health_check():
79     """Health check endpoint."""
80     if not is_model_ready():
81         raise HTTPException(status_code=503, detail="Model not
82             ready")
83     return {"status": "healthy"}
84
85 # Endpoint raiz que muestra informacion basica de la API y los
86 endpoints disponibles
87 @app.get("/")
88 async def root():
89     """Root endpoint with API information."""
90     return {
91         "message": "YOL011 Inference Server",
92         "version": "1.0.0",
93         "endpoints": {
94             "health": AIP_HEALTH_ROUTE,
95             "predict": AIP_PREDICT_ROUTE,
96             "predict_image": "/predict-image",
97             "docs": "/docs"
98         }
99     }

```

```

98
99 @app.post(AIP_PREDICT_ROUTE, response_model=PredictionResponse)
100 async def predict(request: PredictionRequest):
101     """Prediction endpoint."""
102     try:
103         predictions = [] # Lista para almacenar resultados de
104                          # todas las instancias
105
106         # Procesa cada instancia en la request
107         for instance in request.instances:
108             if isinstance(instance, dict):
109                 if "image" in instance:
110                     # Decodifica la imagen de base64
111                     image_data =
112                         base64.b64decode(instance["image"])
113                     # Convierte bytes a imagen PIL/OpenCV
114                     input_image =
115                         get_image_from_bytes(image_data)
116                 else:
117                     raise HTTPException(status_code=400,
118                                         detail="Instance must contain 'image'
119                                               field")
120             else:
121                 raise HTTPException(status_code=400,
122                                     detail="Invalid instance format")
123
124         # Extrae parametros opcionales o usa valores por
125         # defecto
126         parameters = request.parameters or {}
127         confidence_threshold = parameters.get("confidence",
128                                             0.5)

```

```

121     return_annotated_image =
122         parameters.get("return_annotated_image", True)
123
124     # Ejecuta inferencia con el modelo YOLO11
125     result = run_inference(input_image,
126                             confidence_threshold=confidence_threshold)
127     detections_list = result["detections"]
128
129     # Formatea las detecciones para la respuesta
130     detections = []
131     for detection in detections_list:
132         formatted_detection = {
133             "class_name": detection["name"],
134             "confidence": detection["confidence"],
135             "bbox": {
136                 "xmin": detection["xmin"],
137                 "ymin": detection["ymin"],
138                 "xmax": detection["xmax"],
139                 "ymax": detection["ymax"],
140             },
141         }
142         detections.append(formatted_detection)
143
144     # Construye el resultado de prediccion
145     prediction = {
146         "detections": detections,
147         "detection_count": len(detections),
148         "annotated_image": None
149     }
150
151     # Genera imagen anotada si se solicita y hay
152     detecciones

```



```

150         if return_annotated_image and result["results"]:
151             annotated_image =
152                 get_annotated_image(result["results"])
153             img_bytes = get_bytes_from_image(annotated_image)
154             # Codifica la imagen anotada en base64 para la
155                 respuesta
156             prediction["annotated_image"] =
157                 base64.b64encode(img_bytes).decode("utf-8")
158
159             predictions.append(prediction)
160
161             # Log del procesamiento
162             logger.info(
163                 f"Processed {len(request.instances)} instances,
164                     found {sum(len(p['detections']) for p in
165                         predictions)} total detections"
166             )
167
168             return PredictionResponse(predictions=predictions)
169
170     except HTTPException:
171         # Re-lanza HTTPExceptions sin modificar
172         raise
173
174     except Exception as e:
175         # Maneja errores inesperados con log y response 500
176         logger.error(f"Prediction error: {str(e)}")
177         raise HTTPException(status_code=500, detail=f"Prediction
178             failed: {str(e)}")
179
180 if __name__ == "__main__":
181     # Inicializacion del servidor
182     logger.info(f"Starting server on port {PORT}")

```

```

176     logger.info(f"Health check route: {AIP_HEALTH_ROUTE}")
177     logger.info(f"Predict route: {AIP_PREDICT_ROUTE}")
178     # Inicia el servidor uvicorn
179     uvicorn.run(app, host="0.0.0.0", port=PORT)

```

4.4.2 app.py

Listing 4.23: Código app.py

```

1  # Logica de inferencia de YOLO11m
2  from ultralytics import YOLO
3  from PIL import Image
4  import io
5  import os
6  from typing import Dict, Any
7  import cv2
8  import numpy as np
9
10 # Inicializacion del modelo
11 model_yolo = None # Variable global que almacenara el modelo YOLO
12 _model_ready = False # Bandera que indica si el modelo esta
    listo para usar
13
14
15 def _initialize_model():
16     """Inicializar el modelo YOLO."""
17     global model_yolo, _model_ready # Permite modificar las
        variables globales
18
19     try:
20         # Obtener la ruta del modelo desde variables de entorno,
            o usar ruta por defecto
21         model_path = os.getenv("MODEL_PATH", "models/best.onnx")

```

```

22
23     # Verificar si existe un modelo personalizado en la ruta
        especificada
24     if os.path.exists(model_path):
25         print(f"ONNX model loaded from {model_path}")
26         # Cargar modelo YOLO para deteccion
27         model_yolo = YOLO(model_path, task="detect")
28
29     else:
30         # Cargar modelo pre-entrenado YOLO11n
31         print("Pre-trained YOLO11n model loaded")
32         model_yolo = YOLO("yolo11n.pt")
33
34     # Marcar el modelo como listo si se cargo correctamente
35     _model_ready = True
36     print("Modelo iniciado correctamente.")
37
38     except Exception as e:
39         # Manejar cualquier error durante la inicializacion
40         print(f"Error al inicializar modelo YOLO: {e}")
41         _model_ready = False # Marcar como no listo
42         model_yolo = None # Limpiar referencia al modelo
43
44     # Ejecutar la inicializacion al importar el modulo
45     _initialize_model()
46
47     def is_model_ready() -> bool:
48         # Verificar que ambas condiciones sean verdaderas:
49         # - La bandera _model_ready esta en True
50         # - La variable model_yolo no es None
51         return _model_ready and model_yolo is not None
52

```

```

53 def get_image_from_bytes(binary_image: bytes) -> Image.Image:
54     """Convertir imagen de bytes a formato PIL RGB."""
55     # Abrir la imagen y convertir a formato RGB
56     input_image =
57         Image.open(io.BytesIO(binary_image)).convert("RGB")
58     return input_image
59
60 def get_bytes_from_image(image: Image.Image) -> bytes:
61     """Convertir formato imagen PIL a bytes."""
62     # Crear un buffer en memoria para almacenar la imagen
63     return_image = io.BytesIO()
64     # Guardar la imagen en el buffer en formato JPEG con calidad
65     85%
66     image.save(return_image, format="JPEG", quality=85)
67     # Mover el puntero al inicio del buffer para lectura
68     return_image.seek(0)
69     # Obtener los bytes del buffer
70     return return_image.getvalue()
71
72 def run_inference(input_image: Image.Image,
73     confidence_threshold: float = 0.5) -> Dict[str, Any]:
74     """Ejecutar inferencia en una imagen usando el modelo
75     YOLO11."""
76     global model_yolo # Referencia a la variable global del
77     modelo
78
79     # Verificar si el modelo esta listo
80     if not is_model_ready():
81         print("Model not ready for inference")
82         # Retornar estructura vacia si el modelo no esta listo
83         return {"detections": [], "results": None}

```

```

80     try:
81         # Ejecutar prediccion con parametros especificos:
82         results = model_yolo.predict(
83             imgsz=640,          # Tamano de imagen
84             source=input_image,  # Imagen de entrada
85             conf=confidence_threshold, # Umbral de confianza
86             save=False,         # No guardar imagenes
87                                 # automaticamente
88             augment=False,      # No usar aumentacion de datos
89             verbose=False       # No mostrar output verbose en
90                                 # consola
91         )
92
93         # Extraer detecciones (bounding boxes, class names y
94         # confidences)
95         detections = [] # Lista para almacenar todas las
96         # detecciones
97
98         if results and len(results) > 0: # Verificar que hay
99         # resultados
100             result = results[0] # Tomar el primer resultado
101             # (solo una imagen)
102
103             if result.bboxes is not None and
104             len(result.bboxes.xyxy) > 0:
105                 bboxes = result.bboxes # Acceder a las cajas
106                 # delimitadoras
107
108                 # Convertir tensores de PyTorch a arrays de
109                 # numpy para procesamiento
110
111                 xyxy = bboxes.xyxy.cpu().numpy() # Coordenadas
112                 [xmin, ymin, xmax, ymax]
113
114                 conf = bboxes.conf.cpu().numpy() # Niveles de
115                 # confianza

```

```

101         cls = boxes.cls.cpu().numpy().astype(int) #
           Clases (convertidas a enteros)
102
103     # Crear diccionarios con la informacion de cada
           deteccion
104     for i in range(len(xyxy)):
105         detection = {
106             "xmin": float(xyxy[i][0]), # Coordenada
           x minima
107             "ymin": float(xyxy[i][1]), # Coordenada
           y minima
108             "xmax": float(xyxy[i][2]), # Coordenada
           x maxima
109             "ymax": float(xyxy[i][3]), # Coordenada
           y maxima
110             "confidence": float(conf[i]), #
           Confianza de la deteccion
111             "class": int(cls[i]), # ID de la clase
112             # Obtener nombre de la clase del
           diccionario de nombres del modelo
113             "name":
           model_yolo.names.get(int(cls[i]),
           f"class_{int(cls[i])}"),
114         }
115         detections.append(detection) # Agregar
           deteccion a la lista
116
117     # Retornar resultados estructurados
118     return {
119         "detections": detections, # Lista de detecciones
           formateadas
120         "results": results, # Resultados brutos para
           anotaciones posteriores

```

```

120     }
121     except Exception as e:
122         # Retornar estructura vacia en caso de error
123         print(f"Error in YOLO detection: {e}")
124         return {"detections": [], "results": None}
125
126 def get_annotated_image(results: list) -> Image.Image:
127     """Obtener imagen anotada usando metodo plot de
128     Ultralytics."""
129     if not results or len(results) == 0:
130         raise ValueError("Sin resultados para la anotacion")
131     result = results[0] # Tomar el primer resultado
132     # Siempre obtener como numpy array y convertir BGR to RGB
133     # Usar el metodo plot() de Ultralytics que devuelve imagen
134     # anotada en formato BGR
135     annotated_img_bgr = result.plot() # Esto devuelve BGR
136     # Convertir de BGR a RGB (formato PIL)
137     annotated_img_rgb = cv2.cvtColor(annotated_img_bgr,
138                                     cv2.COLOR_BGR2RGB)
139     # Convertir array de numpy a imagen PIL
140     return Image.fromarray(annotated_img_rgb)

```

4.5 Anexo C : Código de la función en Firebase Functions

4.5.1 main.py

Listing 4.24: Codigo main.py

```

1 from firebase_functions import storage_fn
2 from firebase_admin import initialize_app, storage, firestore
3 import requests
4 import io

```

```

5 import logging
6 import re
7 import time
8 import base64
9
10 app = initialize_app()
11 logging.basicConfig(level=logging.INFO)
12 logger = logging.getLogger('cloud_functions')
13
14 @storage_fn.on_object_finalized()
15 def process2_cherry_image(event:
    storage_fn.CloudEvent[storage_fn.StorageObjectData]):
16     file_path = event.data.name
17
18     path_pattern =
19         r'^user/([^/]+)/photos/([^/]+)/([^/]+\.(jpg|jpeg|png))$'
20     match = re.match(path_pattern, file_path)
21
22     if not match:
23         logger.info(f"Ignorando archivo {file_path}")
24         return
25
26     user_id = match.group(1)
27     photo_id = match.group(2)
28     file_name = match.group(3)
29
30     if not file_name.lower().endswith((' .jpg', ' .jpeg', ' .png')):
31         logger.info(f"Ignorando archivo {file_path}")
32         return
33
34     bucket_name = event.data.bucket
35     cloud_run_url = "URL DEL SERIVIO CLOUD"

```



```

35
36     bucket = storage.bucket(bucket_name)
37     original_blob = bucket.blob(file_path)
38
39     try:
40         logger.info(f"Iniciando procesamiento para usuario
41                     {user_id}, foto {photo_id}")
42
43         # Descargar imagen
44         image_bytes = io.BytesIO()
45         original_blob.download_to_file(image_bytes)
46         image_bytes.seek(0)
47
48         image_data = image_bytes.getvalue()
49         image_size = len(image_data)
50         logger.info(f"Tamaño: {image_size} bytes
51                     ({image_size/1024:.2f} KB)")
52
53         if image_size == 0:
54             raise Exception("La imagen descargada esta vacia")
55
56         # REINTENTOS
57         max_retries = 3
58         retry_delay = 5
59
60         for attempt in range(max_retries):
61             try:
62                 logger.info(f"Intento {attempt +
63                             1}/{max_retries}")
64
65                 # Codificar imagen a base64

```

```

63         image_base64 =
64             base64.b64encode(image_data).decode('utf-8')
65
66         # Petición a Cloud Run
67         response = requests.post(
68             cloud_run_url,
69             json={
70                 'instances': [{'image': image_base64}],
71                 'parameters': {
72                     'confidence': 0.5,
73                     'return_annotated_image': True
74                 }
75             },
76             headers={'Content-Type': 'application/json'},
77             timeout=120
78         )
79
80         logger.info(f"Respuesta: {response.status_code}")
81
82         # Verificar respuesta exitosa
83         if response.status_code == 200:
84             json_data = response.json()
85             predictions = json_data.get('predictions',
86                                     [])
87
88             if not predictions:
89                 logger.warning("No se encontraron
90                             predicciones")
91
92                 detections_list = []
93                 detection_count = 0
94                 class_counts = {}
95                 annotated_image_base64 = None

```

```

92         else:
93             prediction = predictions[0]
94             detections =
95                 prediction.get('detections', [])
96             detection_count =
97                 prediction.get('detection_count', 0)
98             annotated_image_base64 =
99                 prediction.get('annotated_image')
100
101             logger.info(f"{detection_count}
102                 detecciones encontradas")
103
104             # Procesar detecciones
105             class_counts = {}
106             detections_list = []
107
108             for detection in detections:
109                 # Buscar class_name (tu API lo envia
110                     asi)
111                 class_name =
112                     detection.get('class_name',
113                         'unknown')
114                 confidence =
115                     detection.get('confidence', 0.0)
116                 bbox = detection.get('bbox', {})
117
118                 # Contar por clase
119                 if class_name in class_counts:
120                     class_counts[class_name] += 1
121                 else:
122                     class_counts[class_name] = 1

```

```

116         # Guardar deteccion
117         detections_list.append({
118             'class_name': class_name,
119             'confidence': confidence,
120             'bbox': bbox
121         })
122
123         logger.info(f"Por clase: {class_counts}")
124
125     # Guardar imagen anotada
126     result_path = None
127     result_url = None
128
129     if annotated_image_base64:
130         result_path =
131             f'resultados/{user_id}/photos/{photo_id}/{file_name}
132         result_blob = bucket.blob(result_path)
133
134         # Decodificar base64
135         annotated_image_bytes =
136             base64.b64decode(annotated_image_base64)
137
138         result_blob.upload_from_string(
139             annotated_image_bytes,
140             content_type='image/jpeg'
141         )
142
143         result_blob.make_public()
144         result_url = result_blob.public_url
145
146         logger.info(f"Imagen guardada:
147             {result_url}")

```

```

145         else:
146             logger.warning("No se genero imagen
                               anotada")
147
148         # Actualizar Firestore
149         photo_doc_ref = (
150             firestore.client()
151             .collection('user')
152             .document(user_id)
153             .collection('photos')
154             .document(photo_id)
155         )
156
157         update_data = {
158             'original_image_path': file_path,
159             'detection_status': 'completed',
160             'processed_at':
161                 firestore.SERVER_TIMESTAMP,
162             'attempts': attempt + 1,
163             'detection_count': detection_count,
164         }
165
166         if result_path and result_url:
167             update_data['processed_image_path'] =
168                 result_path
169             update_data['processed_image_url'] =
170                 result_url
171
172         if detection_count > 0:
173             update_data['detections'] =
174                 detections_list

```

```

171         update_data['class_counts'] =
172             class_counts
173
174         photo_doc_ref.update(update_data)
175
176         logger.info(f"Exito - Usuario: {user_id},
177                     Foto: {photo_id}")
178         logger.info(f"Total: {detection_count}")
179         if class_counts:
180             for cls, count in class_counts.items():
181                 logger.info(f"    - {cls}: {count}")
182
183         return # Salir tras exito
184
185     # Si fallo
186     error_msg = f"Cloud Run: {response.status_code}
187                 - {response.text[:200]}"
188     logger.error(f"{error_msg}")
189
190     if attempt == max_retries - 1:
191         raise Exception(f"Fallo tras {max_retries}
192                         intentos. {error_msg}")
193
194     except requests.Timeout:
195         logger.warning(f"Timeout intento {attempt + 1}")
196         if attempt == max_retries - 1:
197             raise Exception(f"Timeout tras {max_retries}
198                             intentos")
199
200     except requests.exceptions.RequestException as e:
201         logger.warning(f"Error conexion: {str(e)}")
202         if attempt == max_retries - 1:

```

```

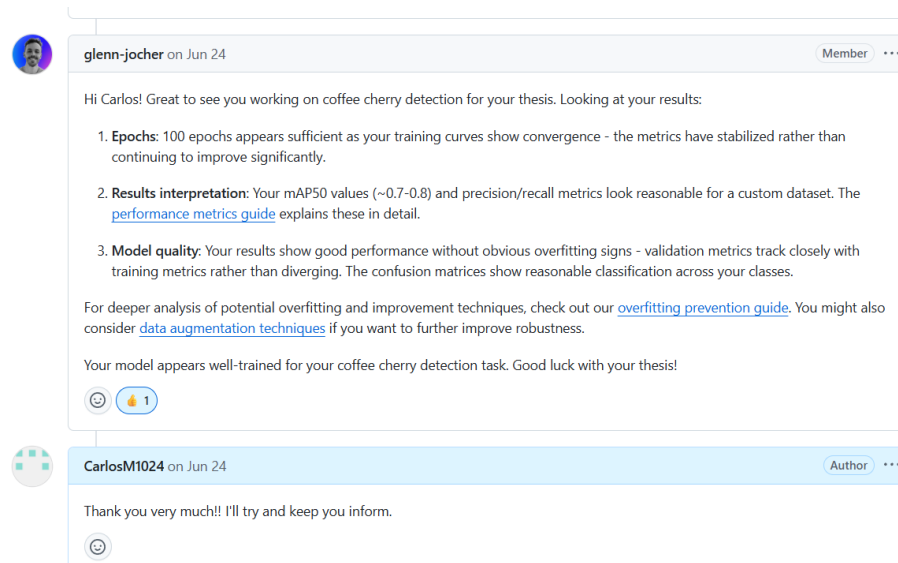
198         raise
199
200         # Esperar antes del siguiente intento
201         if attempt < max_retries - 1:
202             wait_time = retry_delay * (2 ** attempt)
203             logger.info(f"Esperando {wait_time}s...")
204             time.sleep(wait_time)
205
206     except Exception as e:
207         error_msg = f"Error: {str(e)}"
208         logger.error(f"{error_msg} - Usuario {user_id}, Foto
209                     {photo_id}")
210
211         firestore.client().collection('user').document(user_id).collection('ph
212             'detection_status': 'error',
213             'error_message': error_msg,
214             'error_at': firestore.SERVER_TIMESTAMP
215         })
216
217         raise

```

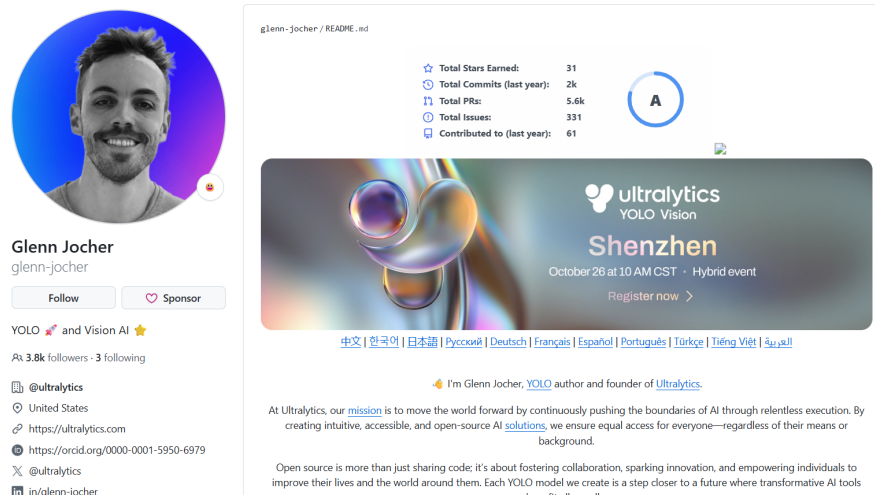
4.6 Anexo D : Asesoría con Glenn Jocher, CEO de Ultralytics

Durante desarrollo del proyecto de investigación, surgieron dudas referentes a los resultados obtenidos en el entrenamiento del modelo YOLO11, por lo que se abrió un ‘issue’ en el repositorio oficial de Ultralytics en GitHub. En este issue, me asesoró Glenn Jocher, CEO de Ultralytics, quien resolvió mis dudas y proporcionó recomendaciones para mejorar el entrenamiento del modelo.

A continuación, se presentan capturas de pantalla de la conversación sostenida en GitHub y el [enlace](#) para ingresar al issue.



(a)



(b)

Figura 71: Capturas de pantalla de GitHub: (a) conversación con Glenn Jocher, CEO de Ultralytics y (b) perfil de Glenn Jocher en GitHub.