DESARROLLO DE SISTEMA DE RECONOCIMIENTO DE FLORA BOGOTANA BASADO EN DEEP LEARNING PARA DISPOSITIVOS MÓVILES

Trabajo de grado para optar el título de Ingeniero de Sistemas y Computación

JUAN DIEGO LIZARAZO PAEZ

UNIVERSIDAD DE CUNDINAMARCA EXTENSIÓN CHÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

FACULTAD DE INGENIERÍA

2025

DESARROLLO DE SISTEMA DE RECONOCIMIENTO DE FLORA BOGOTANA BASADO EN DEEP LEARNING PARA DISPOSITIVOS MÓVILES

JUAN DIEGO LIZARAZO PAEZ

CÓDIGO: 561221240

DIRECTOR

Ing. Esp. Edison Gustavo Cañon Varela

UNIVERSIDAD DE CUNDINAMARCA EXTENSIÓN CHÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

FACULTAD DE INGENIERÍA

2025

AGRADECIMIENTOS

En primer lugar, agradezco a Dios por iluminar mi camino, bendecirme con sabiduría y guiarme en todo el proceso académico. Agradezco de igual manera a mis padres y hermanos por ser apoyo incondicional a lo largo de la carrera, a mi director el profesor Edison Cañón por ayudarme a materializar el proyecto, al Ing. Eduard Gutiérrez por sus gestiones como coordinador del programa, y a todos los buenos docentes que demostraron competencia y pasión por la docencia, y que contribuyeron a mi formación como ingeniero y como persona.

RESUMEN

El fomento a la valoración de la biodiversidad urbana en contextos urbanos se ve limitado por la falta de herramientas que conecten al ciudadano con la información disponible en sistemas especializados. Este proyecto de abordó dicha problemática mediante el desarrollo de un sistema de reconocimiento de la flora bogotana implementado en una aplicación móvil, utilizando técnicas de aprendizaje profundo. Se aplicó el marco metodológico CRISP-ML(Q), comenzando con la construcción de un dataset propio de 4,188 imágenes curadas de 11 especies representativas de Bogotá, obtenidas de plataformas de ciencia ciudadana. Se implementó una estrategia de aprendizaje por transferencia para entrenar y evaluar comparativamente cuatro arquitecturas de Redes Neuronales Convolucionales (CNN): ResNet-50, MobileNetV2, MobileNetV3-Large y EfficientNet-B0. El análisis cuantitativo sobre un conjunto de prueba alcanzó una exactitud del 95.47% y un F1-Score (Macro) del 94.11% en la arquitectura EfficientNetB0. Para el despliegue final en la aplicación móvil, se seleccionó y optimizó el modelo MobileNetV3-Large por su balance superior entre precisión, latencia de inferencia y tamaño reducido. El desarrollo culminó en una aplicación móvil que valida la viabilidad de crear herramientas de alta precisión para el reconocimiento de flora regional.

Palabras clave: Aprendizaje por transferencia, Clasificación de imágenes, Flora bogotana, Redes neuronales convolucionales, Visión por computador

ABSTRACT

Fostering the valuation of urban biodiversity is hampered by the lack of tools that connect citizens with information available in specialized systems. This project addressed this issue by developing a recognition system for the flora of Bogotá, implemented in a mobile application using deep learning techniques. The CRISP-ML(Q) methodological framework was applied, commencing with the construction of a custom dataset of 4,188 curated images from 11 representative species of Bogotá, obtained from citizen science platforms. A transfer learning strategy was implemented to comparatively train and evaluate four Convolutional Neural Network (CNN) architectures: ResNet-50, MobileNetV2, MobileNetV3-Large, and EfficientNet-B0. Quantitative analysis on a test set showed that the EfficientNetB0 architecture achieved a top accuracy of 95.47% and a Macro F1-Score of 94.11%. For the final deployment in the mobile application, the MobileNetV3-Large model was selected and optimized for its superior balance between accuracy, inference latency, and reduced size. The development culminated in a functional mobile app that validates the feasibility of creating high-precision tools for regional flora recognition.

Keywords: Computer Vision, Convolutional Neural Networks, Bogotá Flora, Image Classification, Transfer Learning

TABLA DE CONTENIDO

INTRODUCCIÓN 11

CAPÍTULO 1 12

1. PROBLEMA 12

1.1 Planteamiento del problema 12

1.2 Formulación del problema 13

2. OBJETIVOS 14

2.1 Objetivo general 14

2.2 Objetivos específicos 14

3. ALCANCES Y LIMITACIONES 15

4. JUSTIFICACIÓN 16

5. LÍNEAS DE INVESTIGACIÓN 17

CAPÍTULO 2 18

6. MARCO TEÓRICO 18

6.1 Marco referencial 18

6.2 Marco conceptual 23

6.3 Marco ingenieril 29

CAPÍTULO 3 48

7. METODOLOGÍA 48

8. DESARROLLO DEL PROYECTO 51

8.1 Preparación de datos 51

8.2 Modelado 58

8.3 Evaluación 64

8.4 Despliegue: Optimización del modelo 74

8.5 Despliegue: Aplicación móvil 75

CAPÍTULO 4 86

10. CONCLUSIONES 86

11. RECOMENDACIONES 87

REFERENCIAS 89

LISTA DE FIGURAS

Figura 1: Relación jerárquica entre Inteligencia artificial, Aprendizaje automático y Aprendizaje profundo 25
Figura 2: Jerarquía de características en una red neuronal convolucional 27
Figura 3: Bloque de construcción de MobileNetV2 con residuo invertido 31
Figura 4: Bloque de construcción de MobileNetV3 32
Figura 5: Comparativa de funciones de activación soft y hard 33
Figura 6: Comparativa de subajuste, ajuste óptimo y sobreajuste 37
Figura 7: Ejemplo de transformación de data augmentation. 40
Figura 8: Arquitectura de React Native 45
Figura 9: Arquitectura de la aplicación móvil 47
Figura 10: Fases del proceso de desarrollo del modelo CRISP-ML(Q) 48
Figura 11: Diagrama de actividad del flujo de entrenamiento del modelo FloraBogNet 62
Figura 12: Evolución de la exactitud en el conjunto de validación durante el entrenamiento 65
Figura 13: Evolución de la función de pérdida en el conjunto de validación durante el entrenamiento 66
Figura 14: Curvas de exactitud en entrenamiento y validación por arquitectura 67
Figura 15: Curvas de exactitud en entrenamiento y validación por arquitectura 68
Figura 16: Matriz de confusión de EfficientNetB0 sobre el conjunto de prueba 69
Figura 17: Visualización Grad-CAM para las arquitecturas evaluadas sobre una misma imagen 71
Figura 18: Visualización Grad-CAM para EfficientNetB0 evaluada sobre imágenes de varias especies 72
Figura 19: Visualización Grad-CAM para EfficientNetB0 evaluada sobre diferentes partes de la misma especie 73
Figura 20: Diagrama de caso de uso CU-01: Identificación automatizada de especies 76
Figura 21: Diagrama de caso de uso CU-02: Exploración de catálogo de flora 77
Figura 22: Diagrama de caso de uso CU-03: Gestión de observaciones personales 78
Figura 23: Wireframe de alta fidelidad para app móvil - pantallas principales 79
Figura 24: Wireframe de alta fidelidad para app móvil - detalle de especie 80
Figura 25: Diagrama de actividad - flujo de desarrollo y lanzamiento de la aplicación móvil 82

LISTA DE TABLAS

Tabla 1: Especies seleccionadas para el modelo 51
Tabla 2: Distribución de imágenes de grado de investigación descargadas por especie 53
Tabla 3: Distribución de imágenes con identificación necesaria descargadas por especie..... 55
Tabla 4: Distribución final de imágenes por especie 56
Tabla 5: Comparativa de rendimiento de arquitecturas entrenadas en el conjunto de prueba 64
Tabla 6: Análisis comparativo del rendimiento post-cuantización de las arquitecturas MobileNetB0 y EfficientNet-B3. 74
Tabla 7: Dispositivos físicos utilizados en la fase de pruebas 83

ACRÓNIMOS

- API: Interfaz de programación de aplicaciones
- CI/CD: Integración continua y entrega continua
- CNN: Red neuronal convolucional
- CPU: Unidad central de procesamiento
- CRISP-DM: Proceso estándar interindustrial para minería de datos (Cross-Industry Standard Process for Data Mining en inglés).
- CRISP-ML(Q): Proceso estándar interindustrial para aprendizaje automático con aseguramiento de la calidad (Cross-Industry Standard Process for Machine Learning with Quality Assurance en inglés).
- CSV: Formato de archivo de valores separados por comas
- DVC: Control de versiones de datos
- FLOPS: Operaciones de punto flotante por segundo
- GBIF: Sistema Global de Información sobre Biodiversidad (Global Biodiversity Information Facility en inglés).
- GPU: Unidad de procesamiento gráfico
- Grad-CAM: Mapa de activación de clases ponderado por gradientes
- IA: Inteligencia artificial
- JBB: Jardín Botánico de Bogotá José Celestino Mutis
- ML: Aprendizaje automático
- NAS: Búsqueda de arquitectura neuronal
- POT: Plan de ordenamiento territorial
- SDP: Secretaría Distrital de Planeación
- SiB Colombia: Sistema de Información sobre Biodiversidad de Colombia
- SIGAU: Sistema de Información para la Gestión del Arbolado Urbano
- SVM: Máquina de vectores de soporte
- TFLite: TensorFlow Lite
- UI/UX: Interfaz de usuario y experiencia de usuario
- XAI: Inteligencia artificial explicable

INTRODUCCIÓN

La biodiversidad urbana constituye un elemento esencial para el sostenimiento ecológico de las ciudades y para la preservación de sus servicios ecosistémicos. A pesar de lo anterior, en contextos metropolitanos como el de la ciudad de Bogotá persiste una desconexión entre la ciudadanía y la flora local, derivada de la dificultad para acceder, comprender y apropiarse del conocimiento de biodiversidad disponible en sistemas especializados.

El presente trabajo de grado plantea como objetivo desarrollar un sistema de reconocimiento automático de especies de flora bogotana basado en técnicas de Deep Learning, desplegado en una aplicación móvil, buscando ofrecer una herramienta que facilite la identificación *in situ* de especies, fomentando tanto el aprendizaje ciudadano como la conservación de la biodiversidad urbana. La hipótesis central se sustenta en que el uso de modelos de visión por computador entrenados con datos locales puede superar las limitaciones de aplicaciones globales existentes, generando un mecanismo adaptado al contexto específico de Bogotá.

La investigación se apoya en la revisión de antecedentes sobre biodiversidad urbana, sistemas de información taxonómica y aplicaciones previas de identificación vegetal, así como en el análisis de arquitecturas de redes neuronales convolucionales optimizadas para dispositivos móviles. La metodología seguida se estructura a partir del modelo CRISP-ML(Q), abarcando las fases de recolección y curación de un dataset local, el entrenamiento y evaluación de arquitecturas mediante aprendizaje por transferencia, y el posterior despliegue del modelo en una aplicación móvil.

CAPÍTULO 1

1. PROBLEMA

1.1 Planteamiento del problema

La biodiversidad, entendida como la variedad de especies que coexisten en un ecosistema, es un elemento fundamental para la sostenibilidad y el equilibrio ecológico de las ciudades. Sin embargo, la percepción y concientización ciudadana sobre la conservación esta biodiversidad tiende a ser limitada. Es esencial considerar a las poblaciones humanas como parte integral de los sistemas socio-ecológicos, lo cual implica que su valoración de la biodiversidad y sus servicios ecosistémicos influye directamente en la conservación (López Barrera et al., 2017). Aun así, en muchos entornos urbanos ha desaparecido la noción de la interdependencia entre las actividades humanas y el medio ambiente.

Bogotá, como una de las capitales con mayor biodiversidad en su área de influencia enfrenta una disociación socioambiental donde los habitantes perciben el entorno natural como un elemento ajeno, olvidando que la sostenibilidad urbana depende intrínsecamente de los recursos naturales (Leiva & Rodríguez , 2021, p. 5). Lo anterior se traduce en valoración ciudadana de la flora local sin herramientas, y a su vez en obstáculo para los esfuerzos de concientización, difusión y apropiación del conocimiento.

Frente a lo anterior, se encuentran instituciones de alcance distrital y nacional que han desarrollado herramientas de grado académico, como: el proyecto Flora de Bogotá con el Catálogo de la flora vascular de Bogotá (Jardín Botánico de Bogotá José Celestino Mutis [JBB], 2024), el Sistema de Información para la Gestión del Arbolado Urbano – SIGAU (JBB, 2023), el Herbario Virtual (JBB, 2019), y el Catálogo de Plantas y Líquenes de Colombia compilado por Raz & Zamora (2023) para la Universidad Nacional de Colombia. Estas herramientas concentran una vasta cantidad de información taxonómica y de gestión, pero que, por su naturaleza académica y científica están diseñadas principalmente como sistemas de consulta, exigiendo que el usuario posea un conocimiento previo para navegar, acceder, consumir y analizar los datos de manera efectiva.

La problemática central que este proyecto aborda radica en tal brecha de accesibilidad. Para el ciudadano no experto, la tarea de identificar una especie de planta *in situ* se convierte en un desafío que desincentiva la curiosidad, el aprendizaje y la apropiación. La ausencia de un mecanismo de identificación automatizado y de fácil uso en las herramientas oficiales crea una barrera entre la información existente y el público general.

Si bien podría argumentarse que esta brecha tecnológica es cubierta por una variedad de aplicaciones comerciales y de acceso general para identificar plantas, una evaluación crítica de su desempeño revela limitaciones que justifican el desarrollo de una solución localizada. Estudios como el de Campbell et al. (2023), que analizaron de forma sistemática seis de las aplicaciones más populares (entre ellas iNaturalist, PlantNet y Google Lens), demuestran que su fiabilidad es variable. Se encuentra una "considerable variación" en la precisión de la identificación entre distintas especies y condiciones, concluyendo que ninguna aplicación debe ser "asumida como correcta". Este mismo estudio reportó que incluso las aplicaciones de mayor rendimiento no superaron un techo de precisión de aproximadamente el 88%, lo que implica una tasa de error superior al 12%. Dicho margen de error, aunque es tolerable para la complejidad tecnológica que implica, evidencia la oportunidad de mejora que puede brindar el desarrollo de una herramienta más personalizada. Además, un modelo entrenado con un corpus de datos global puede tener dificultades para diferenciar especies endémicas del altiplano cundiboyacense o variedades locales, que son precisamente las de mayor interés para la conservación y el fomento de la identidad territorial. Por consiguiente, las herramientas globales, aunque útiles como primer acercamiento, no constituyen una solución fiable y adaptada contextualmente, dando cuenta de la necesidad de un sistema más personalizado, entrenado con datos locales y curados, para abordar con precisión la biodiversidad de Bogotá.

1.2 Formulación del problema

La presente investigación se inscribe en línea con la problemática expuesta, guiada de la siguiente pregunta: ¿Cómo diseñar, entrenar y desplegar un sistema basado en Deep Learning para el reconocimiento automático de la flora bogotana, a través de una aplicación móvil que funcione como una herramienta precisa, eficiente y educativa para fomentar la valoración e identificación de la biodiversidad local?

2. OBJETIVOS

2.1 Objetivo general

Desarrollar un sistema de reconocimiento de imágenes basado en Deep Learning, implementado en una aplicación móvil, para la identificación automática de especies de flora bogotana y el fomento de la valoración y apropiación del conocimiento del patrimonio natural de la ciudad.

2.2 Objetivos específicos

- Construir un dataset de imágenes de especies seleccionadas de flora bogotana, aplicando técnicas de aumento de datos para asegurar la calidad y cantidad de muestras necesarias para el entrenamiento del modelo de reconocimiento.
- Entrenar diferentes arquitecturas de redes neuronales convolucionales mediante técnica de transferencia por aprendizaje.
- Evaluar las arquitecturas entrenadas mediante sus métricas de rendimiento y precisión para determinar el modelo óptimo de clasificación de flora.
- Desarrollar una aplicación móvil donde se implemente el modelo de reconocimiento, que permita al usuario realizar clasificaciones de flora en tiempo real y aprender interactivamente sobre la flora de Bogotá.

3. ALCANCES Y LIMITACIONES

El presente proyecto contempla el diseño, entrenamiento y despliegue de un sistema de reconocimiento de 11 especies de flora bogotana basado en técnicas de aprendizaje profundo. En el plano académico, su alcance reside en la construcción de un marco metodológico que integra la revisión de fuentes de información taxonómica, la curación de un conjunto de datos locales y la evaluación comparativa de arquitecturas de redes neuronales convolucionales mediante aprendizaje por transferencia. En el plano tecnológico, el alcance se concreta en la implementación de un modelo optimizado y funcional en una aplicación móvil que identifique en tiempo real el conjunto de especies, con posibilidad de guardar esas observaciones. La aplicación móvil tendrá, aparte de tales funcionalidades, la de explorar un catálogo de especies de la flora bogotana.

En cuanto a las limitaciones del modelo, el entrenamiento de los modelos se ve condicionado por dos factores: la capacidad de hardware disponible, debido a que se necesitan instancias con unidades de procesamiento gráfico (GPU) para el entrenamiento, las cuales tienen un costo incrementado comparado a instancias estándares. Para este tipo de tareas, la GPU debería tener al menos 16GB de VRAM, 12GB de RAM y al menos 20GB de almacenamiento. La otra limitación es la cantidad de datos de calidad que se puedan adquirir para el entrenamiento del modelo, lo cual condicionará la cantidad de especies que se puedan reconocer. Un último aspecto para considerar es que el proyecto se llevó a cabo por una sola persona y esto hace que los resultados se puedan tardar más en materializar a diferencia de un equipo de más personas.

4. JUSTIFICACIÓN

El desarrollo del presente proyecto de ingeniería se justifica a partir de su valor multidimensional. Los avances en Inteligencia Artificial -que se han visto incrementados de manera exponencial en los últimos años- particularmente en el campo de la Visión por Computador, presentan una oportunidad tecnológica para solventar esta problemática. La identificación de especies vegetales es un problema complejo de clasificación de imágenes de grano fino (Wei et al., 2021), donde las Redes Neuronales Convolucionales (CNN) han demostrado una capacidad superior para extraer y diferenciar características sutiles que son imperceptibles para el ojo no entrenado (Muñoz & Bolt, 2021). La metodología de Aprendizaje por Transferencia (Transfer Learning) permite, además, adaptar modelos de alto rendimiento, pre-entrenados en grandes datasets como ImageNet, a dominios específicos como la flora local, logrando resultados precisos con una inversión computacional factible (Gómez Gómez et al., 2021; Krishna et al., 2020).

Al proponer la creación de una herramienta tecnológica resultante que convierte datos taxonómicos, actualmente confinados en catálogos y bases de datos especializadas en conocimiento accesible, inmediato e interactivo. Su principal utilidad radica en la capacidad de realizar identificaciones in situ, eliminando la barrera de entrada que el conocimiento experto previo representa y transformando el proceso de consulta pasiva en una utilidad descubrimiento interactivo. En el ámbito educativo se concibe como estrategia de aprendizaje móvil situado “M-Learning”, que ocurre en un contexto auténtico y se ve potenciado al proporcionar pistas y estructura para guiar al estudiante fuera de los entornos habituales de aprendizaje (Naveed et al., 2023).

De igual manera, el proyecto contribuye socialmente al alinearse con las estrategias de gobernanza ambiental y fomentar la ciencia ciudadana. Como lo subraya Quimbayo-Ruiz (2016) en el marco de trabajo del Instituto Humboldt, existe la necesidad de fortalecer la relación entre el Estado y la ciudadanía a través del reconocimiento del valor público de la biodiversidad. Este proyecto atiende directamente a ese llamado, ofreciendo un canal tecnológico para materializar dicha conexión. La aplicación se inserta en un proceso que, como expone el mismo autor, “viene gestándose desde hace más de dos décadas a partir de la acción de múltiples actores, organizaciones y ciudadanos” (p. 61). De este modo, la herramienta propuesta tiene el potencial

de involucrar a nuevos públicos y de facilitar la labor de los actores sociales consolidados con la causa ambiental en la ciudad. A mediano plazo, la base de código y el modelo de reconocimiento pueden ser extendidos con más especies de la zona, y también replicados para ser adaptadas a otras zonas del país con objetivos similares.

5. LÍNEAS DE INVESTIGACIÓN

El presente proyecto se inscribe dando cumplimiento con el Acuerdo No. 009 de 2021 establecido por la Universidad de Cundinamarca (2021) en la línea translocal de investigación denominada “Transmodernidad, naturaleza, ambiente, biodiversidad, ancestralidad y familia” y la línea complementaria de “Software, sistema emergentes y nuevas tecnologías” correspondiente a la Facultad de Ingeniería.

CAPÍTULO 2

6. MARCO TEÓRICO

6.1 Marco referencial

En el presente apartado se expone de manera contextualizada los antecedentes que son objeto de estudio en el presente proyecto.

6.1.1 Biodiversidad urbana en Bogotá. El Distrito Capital, con una extensión de 163.660 hectáreas (23,41% área urbana y 76,59% área rural), alberga una diversidad abundante de ecosistemas que asciende a más de 90 tipos rurales y a más de 400 unidades ambientales urbanas, donde habitan más de 600 especies de flora y potencialmente más de 200 especies de fauna (Secretaría Distrital de Planeación [SDP], 2021).

Este panorama de biodiversidad urbana está caracterizado por una interacción entre elementos naturales y antropizados que configuran lo que algunos autores como Leiva & Rodriguez (2021) denominan "biodiversidad sinantrópica". Específicamente, la flora silvestre urbana comprende aproximadamente 348 especies de plantas vasculares que crecen de forma espontánea sin intervención humana directa, representando desde diminutas hierbas hasta grandes árboles que han logrado adaptarse a la matriz urbana.

Una de las normativas principales que contribuye en la regulación, planificación y gestión del aspecto de biodiversidad es el Plan de Ordenamiento Territorial (POT), dentro del cual, entre otras disposiciones, se establece la Estructura Ecológica Principal (EEP) que constituye el eje vertebrador de la biodiversidad urbana bogotana. Esta red de espacios y corredores que sostiene y conduce los procesos ecológicos esenciales cubre una superficie de 96.727 hectáreas -equivalentes al 59,1% del Distrito Capital- (Decreto 555 de 2021). De igual manera, el POT introduce la figura de los conectores ecosistémicos para asegurar el flujo de especies y mantener los servicios ambientales, definiendo cinco de estos conectores que integran áreas estratégicas de la ciudad y su periferia.

La gestión de la biodiversidad en el distrito también se encuentra enmarcada dentro de la Política Pública para la Gestión de la Conservación de la Biodiversidad de la Secretaría Distrital de Ambiente (2022), cuyo plan de acción reconoce que la principal amenaza para la integridad de

la EPP es la continua transformación de los ecosistemas, fenómeno que es impulsado por varias causas como la proliferación de especies exóticas invasoras que desplazan la flora nativa, y el tráfico ilegal de fauna y flora -problema del cual Bogotá es un punto neurálgico a nivel nacional-. Se halla de manera transversal un factor que agrava estas problemáticas, y que es explícitamente reconocido por el plan de acción, el cual es la brecha en el conocimiento detallado sobre la biodiversidad local. La política señala la necesidad de avanzar en el monitoreo de grupos taxonómicos poco estudiados, caracterizar la flora de espacios privados y generar cartografía de ecosistemas a escalas más finas como prerrequisitos para una gestión que permita tomar decisiones informada (pp. 8 – 12).

Teniendo en cuenta lo anterior, el desarrollo de una herramienta tecnológica para la identificación de flora adquiere pertinencia estratégica al alinearse directamente con tal política pública y su línea de acción de Ciencia, Tecnología e Innovación (CT&I), las cuales priorizan la recuperación de saberes y la divulgación de datos sobre la riqueza biológica de Bogotá y sus servicios ecosistémicos. (p. 33). El resultado propuesto por el presente proyecto actúa entonces como mecanismo de materialización al funcionar como un puente tecnológico entre los sistemas de conocimiento científico y la divulgación de conocimiento.

6.1.2 Sistemas y herramientas existentes de gestión de flora y taxonomía. El desarrollo de herramientas digitales para la gestión y consulta de información taxonómica ha sido un proceso evolutivo que ha transitado desde sistemas especializados hasta plataformas integradas de acceso global. En el ámbito académico y científico, se encuentra el desarrollado de varias categorías de herramientas que van desde bases de datos taxonómicas hasta sistemas expertos para identificación.

A nivel global y nacional se encuentran sistemas como el Global Biodiversity Information Facility (GBIF), una red internacional que proporciona acceso libre y abierto a datos de presencia de especies de miles de colecciones de museos, herbarios e instituciones de investigación de todo el mundo (GBIF, s/f). El Sistema de Información sobre Biodiversidad de Colombia (SiB Colombia) funciona como el nodo nacional de esta red, articulando los datos del país con la comunidad científica global (SiB Colombia, s/f). Estos portales están diseñados como herramientas para investigadores y gestores de datos, requiriendo un conocimiento técnico y taxonómico para formular consultas e interpretar los resultados.

Otra de las iniciativas académicas es World Flora Online (WFO), una plataforma gestionada por redes de expertos taxonómicos (TENs) que busca completar una flora digital de todas las plantas conocidas del mundo (Kindt, 2020) . Esta plataforma se limita a presentar la información taxonómica, distribución nativa y más información sistematizada, pero sin muestras fotográficas que permitan asociar visualmente la planta.

A nivel distrital, se destaca al Jardín Botánico de Bogotá José Celestino Mutis como entidad que ha desarrollado herramientas especializadas que representan el estado del arte en materia de información taxonómica y gestión de la biodiversidad urbana dentro de la capital. Se destacan las siguientes herramientas:

- El Herbario JBB en línea constituye una biblioteca virtual especializada en la flora de las regiones andina y paramuna, posee más de 20,000 ejemplares de plantas vasculares agrupados en 221 familias, 1,284 géneros y 3,324 especies y ofrece imágenes en alta resolución de ejemplares botánicos en fichas de herbario (JBB, 2019).
- El Sistema de Información para la Gestión del Arbolado Urbano (SIGAU), que centraliza los datos georreferenciados del inventario arbóreo en espacio público de la ciudad, y permite conocer características específicas y localización de cada árbol individual, realizar consultas y obtener indicadores estadísticos de la base de datos. Derivada de este sistema, se encuentra la aplicación ArbolApp, que representa un intento de democratización para acceder a información, tanto resumida como detallada sobre el patrimonio arbóreo de la ciudad (JBB, 2023).
- El Catálogo de la Flora Vascular de Bogotá, desarrollado como parte del proyecto Flora de Bogotá, documenta sistemáticamente las especies de plantas vasculares presentes en el Distrito Capital, proporcionando información taxonómica, distribución y características ecológicas. Su presentación es en formato de catálogo científico (JBB, 2024).

El denominador común de las anteriores herramientas (tanto globales como locales) es su diseño como sistemas de consulta o repositorios de información, donde se presupone que el usuario posee un conocimiento taxonómico previo para navegar las interfaces y realizar búsquedas o procesamiento, lo cual, si bien garantiza el rigor científico, constituye una barrera de accesibilidad significativa para el ciudadano no especializado. Otra consideración se toma en cuenta a partir de

los hallazgos de Schrumpf et al. (2024) señalan que, si bien se procura un consenso general, la coexistencia de múltiples redes, sistemas y bases de datos taxonómicas no asegura consistencia porque que cada una utiliza criterios diferentes para determinar qué nombres de especies son aceptados.

6.1.3 Aplicaciones existentes de identificación visual de especies de flora. El panorama de aplicaciones móviles para identificación de plantas ha tenido variaciones exponenciales en la última década. Entre otras, las aplicaciones más estudiadas y usadas son: PlantNet, iNaturalist, Google Lens, Flora Incognita, PictureThis, PlantSnap y LeafSnap.

Dentro de los estudios se encuentra recurrencia en la evaluación de estas aplicaciones dentro de regiones específicas, y dan cuenta del alcance, capacidades y limitaciones cambian por diversos factores. Shapovalov et al. (2019) encontraron que Google Lens logró una precisión del 92.6% en plantas ucranianas, aunque la aplicación mostró dificultades para reconocer especies endémicas locales. Pärtel et al. (2021) encontraron que Flora Incognita alcanzó una precisión del 85.3% en condiciones de campo en Estonia, y que la presencia de órganos reproductivos o la presencia de solo la especie objetivo en el foco mejoraron significativamente el éxito de identificación, concluyendo que las aplicaciones tienen mejor rendimiento identificando fotos con buena composición donde se centre al sujeto – la planta o sus partes. –

Campbell et al. (2023) realizaron uno de los estudios completos en región delimitada, evaluando seis aplicaciones populares con 857 imágenes de 277 especies del Reino Unido, demostrando que PlantNet y LeafSnap obtuvieron el mejor rendimiento, aunque ninguna aplicación superó el 88% de precisión. Un estudio paralelo realizado en plantas herbáceas irlandesas confirmó que PlantNet fue la aplicación con mejor desempeño, identificando correctamente el 88.21% de imágenes florales y 80.36% de hojas a nivel de especie. Por la naturaleza evolutiva y contributiva de las aplicaciones, se encuentra como aspecto notable una mejora en precisión a lo largo del tiempo. Jones & Jones (2025) documentaron mejoras promedio de 20 puntos porcentuales en todas las aplicaciones evaluadas durante un período de tres años (2020-2023), atribuidas principalmente al crowdsourcing continuo de imágenes, donde se destaca a Flora Incognita que ha expandido su capacidad de 4,851 especies en versiones anteriores a aproximadamente 16,000 especies mundialmente, con mejoras en precisión que alcanzan casi el 100% para ciertas especies con las imágenes adecuadas.

Aunque se halla una variedad de estudios en la revisión, ni en el contexto colombiano ni el bogotano se han hecho comparaciones y/o evaluaciones documentadas sobre las aplicaciones comerciales mencionadas con las especies propias del territorio.

6.1.4 Antecedentes técnicos en reconocimiento de flora por visión computacional. La identificación automatizada de especies vegetales mediante visión computacional constituye un campo de estudio consolidado con una significativa trayectoria investigativa que ha transitado desde métodos clásicos hacia el dominio del aprendizaje profundo.

Antes de la hegemonía de las técnicas de aprendizaje profundo, el paradigma predominante se basaba en la extracción manual de características diseñadas por expertos. Una revisión sistemática de 120 estudios publicados entre 2005 y 2015 reveló que la investigación se centraba en la cuantificación de descriptores visuales específicos como la forma, la textura, el color y la estructura de las hojas, los cuales se utilizaban posteriormente para entrenar clasificadores de aprendizaje automático tradicionales como las máquinas de vectores de soporte (SVM) o k-vecinos más cercanos (k-NN) (Wäldchen & Mäder, 2018). Se demostró la viabilidad del reconocimiento automático, pero dependía mucho de imágenes capturadas en condiciones de laboratorio y no se aseguraba fiabilidad en escenarios no controlados.

La última década ha sido testigo de un cambio de paradigma impulsado por la madurez de las redes neuronales convolucionales (CNN), que automatizan el proceso de extracción de características directamente desde los píxeles de la imagen, estableciendo la tarea como un problema de clasificación visual de grano fino (fine-grained visual classification), donde el objetivo es discernir entre categorías visualmente muy similares (Pérez Esteve, 2020).

Dentro del dominio del aprendizaje profundo, la estrategia metodológica dominante en la actualidad es el aprendizaje por transferencia *(transfer learning)*, que adapta arquitecturas de CNN pre-entrenadas en datasets a gran escala para tareas específicas, y que da como resultado la reducción en el tiempo de entrenamiento, la disminución en dependencia de enormes volúmenes de datos y permite el uso de arquitecturas que pueden aumentar en complejidad (Krishna et al., 2020). La efectividad de este enfoque ha sido ampliamente validada; por ejemplo, aplicando este método con una arquitectura VGG-19 sobre el Swedish Leaf Dataset, compuesto por 15 especies de árboles, se ha logrado una precisión de clasificación del 99.70%, demostrando un rendimiento

cercano a la perfección en condiciones controladas (Siddharth et al., 2022). De igual manera, investigaciones más recientes que proponen arquitecturas de CNN compactas y personalizadas han alcanzado precisiones superiores al 99% en datasets de referencia como PlantVillage y Flavia (Wagle et al., 2021).

Con el ánimo de mejorar la identificación de la flora endémica y nativa de regiones geográficas específicas, se encontraron trabajos que desarrollaron sistemas especializados para flora local, donde se prioriza la curación de datasets propios, a menudo capturados con dispositivos móviles en condiciones de campo. Se encontraron varios ejemplos notables: la creación de una aplicación móvil para la flora nativa de Chile, donde a partir de un dataset propio de 6775 imágenes para 46 especies se alcanzó una precisión del 95% mediante un ensamble de modelos basado en la arquitectura MobileNet (Muñoz & Bolt, 2021) el desarrollo de clasificadores para 14 especies arbóreas en el entorno urbano de Cuenca, Ecuador, que alcanzó una exactitud del 91.2% utilizando la arquitectura ResNetV2-101 (Pacheco-Prado et al., 2023); y el reconocimiento de flores endémicas en la región de Misantla, México, donde se reportaron precisiones variables entre 88% y 99% según las condiciones ambientales de la imagen (Flores et al., 2024).

Los anteriores hallazgos dan cuenta que parte de las investigaciones contemporáneas en el campo del reconocimiento de especies se centra en la creación de soluciones especializadas para flora local y en la optimización de los modelos para su despliegue en dispositivos móviles, por consiguiente, se valida la viabilidad y pertinencia de desarrollar un sistema de reconocimiento de flora delimitada a la región de Bogotá.

6.2 Marco conceptual

6.2.1 Inteligencia Artificial. Se entiende por Inteligencia Artificial el campo general que engloba los esfuerzos por crear sistemas capaces de realizar tareas que tradicionalmente requieren inteligencia humana. Las definiciones tienden a agruparse en dos grandes enfoques: aquellos que se centran en la emulación del razonamiento humano y aquellos que se enfocan en la imitación de la acción humana (Berzal, 2018). Una definición pragmática es la propuesta por Rich, quien la describe como "el estudio de cómo hacer que los ordenadores hagan cosas que, por ahora, los humanos hacemos mejor" (Rich, 1983, citado en Berzal, 2018, p. 10). La frontera de lo que se

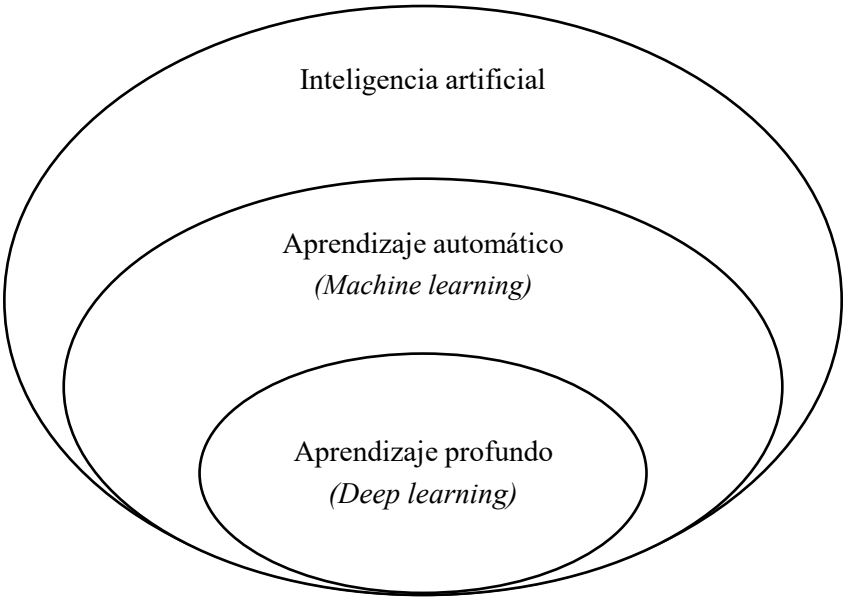
considera IA se desplaza a medida que la tecnología avanza, especialmente en los últimos años, por lo que última es una definición que se ajusta al dinamismo inherente a este campo.

6.2.2 Aprendizaje Automático (Machine Learning). El aprendizaje automático es un subcampo de la Inteligencia Artificial, y aborda el problema de cómo lograr un comportamiento inteligente sin programación explícita, esto es, en lugar de codificar reglas predefinidas, el aprendizaje automático "proporciona mecanismos mediante los cuales el ordenador es capaz de aprender por sí mismo a resolver un problema" (Berzal, 2018, p. 37). Según Samuel (1959, como se citó en Berzal, 2018), esta disciplina faculta a los ordenadores para aprender de los datos, mejorando su rendimiento en una tarea específica a medida que aumenta su experiencia o exposición a la información (p. 37). Formalmente, un sistema de aprendizaje automático mejora su rendimiento (P) en una clase de tareas (T) a través de la experiencia (E) (Mitchell, 1997, como se citó en Goodfellow et al., 2016, p. 99). Adaptado al actual estudio, para la clasificación de flora la tarea (T) es identificar la especie de una planta, la experiencia (E) es el conjunto de imágenes de flora previamente etiquetadas, y el rendimiento (P) se mide a través de la precisión del modelo en la clasificación de nuevas imágenes.

6.2.3 Aprendizaje Profundo (Deep Learning). El aprendizaje profundo es, a su vez, una técnica específica dentro del aprendizaje automático, donde su propósito distintivo es permitir que los computadores "aprendan de la experiencia y comprendan el mundo en términos de una jerarquía de conceptos, donde cada concepto se define en relación con conceptos más simples" (Goodfellow et al., 2016, p. 1). Tal jerarquía se materializa mediante el uso de redes neuronales profundas, compuestas por múltiples capas que automatizan por completo la ingeniería de características (Berzal, 2018). Contrario a los enfoques simbólicos que se fundamentan en reglas lógicas explícitas, las redes neuronales codifican la información de manera implícita mediante el ajuste de pesos en las conexiones sinápticas. Este mecanismo de procesamiento distribuido otorga al sistema una capacidad superior de generalización, permitiéndole realizar inferencias correctas ante patrones que presentan variaciones respecto a los datos de entrenamiento, superando así las limitaciones de desempeño que presentan otros algoritmos convencionales en escenarios complejos (Berzal, 2018, p. 141).

Figura 1

Relación jerárquica entre Inteligencia artificial, Aprendizaje automático y Aprendizaje profundo



Nota. El diagrama de Venn ilustra cómo el Deep learning es un subcampo del Machine learning, que a su vez es una disciplina dentro de la Inteligencia artificial. Adaptado de *Redes Neuronales & Deep Learning* (p. 141), por F. Berzal, 2018.

6.2.4 Visión por computador (Computer Vision). La visión por computador es un campo de la inteligencia artificial que tradicionalmente ha sido una de las áreas de investigación más activas para las aplicaciones de aprendizaje profundo, teniendo como objetivo central el de desarrollar sistemas que puedan procesar, analizar y comprender el contenido de imágenes digitales de manera similar a como lo hace la visión humana. De igual manera, es un campo en el que se materializa la definición de IA expuesta anteriormente, ya que la percepción visual "es una tarea que resulta sencilla para los humanos y muchos animales, pero desafiante para los computadores" (Ballard et al., 1983, citado en Goodfellow et al., 2016, p. 452).

La mayoría de las aplicaciones de aprendizaje profundo en este campo se centran en el reconocimiento o detección de objetos en alguna de sus formas, ya sea indicando qué objeto está presente en una imagen, anotando dicho objeto con un cuadro delimitador, o etiquetando cada píxel de la imagen con la identidad del objeto al que pertenece (Goodfellow et al., 2016). Debido a la

complejidad inherente de los datos visuales, donde factores como la iluminación, la perspectiva, la oclusión y la deformación introducen una inmensa variabilidad, el aprendizaje profundo y, en particular, las redes neuronales convolucionales, se han convertido en el enfoque de vanguardia para resolver estos problemas.

6.2.5 Redes Neuronales Convolucionales (CNN). Las Redes neuronales convolucionales son una clase especializada de redes neuronales diseñadas para procesar datos que poseen una topología de tipo rejilla, como es el caso de las imágenes, que pueden ser representadas como una matriz bidimensional de píxeles (Goodfellow et al., 2016). Su rotundo éxito en la resolución de problemas de visión artificial ha sido el principal catalizador de la revolución del aprendizaje profundo (Berzal, 2018). La diferencia fundamental de las CNN con respecto a las redes neuronales multicapa tradicionales reside en que, en al menos una de sus capas, reemplazan la multiplicación matricial general por una operación matemática denominada convolución.

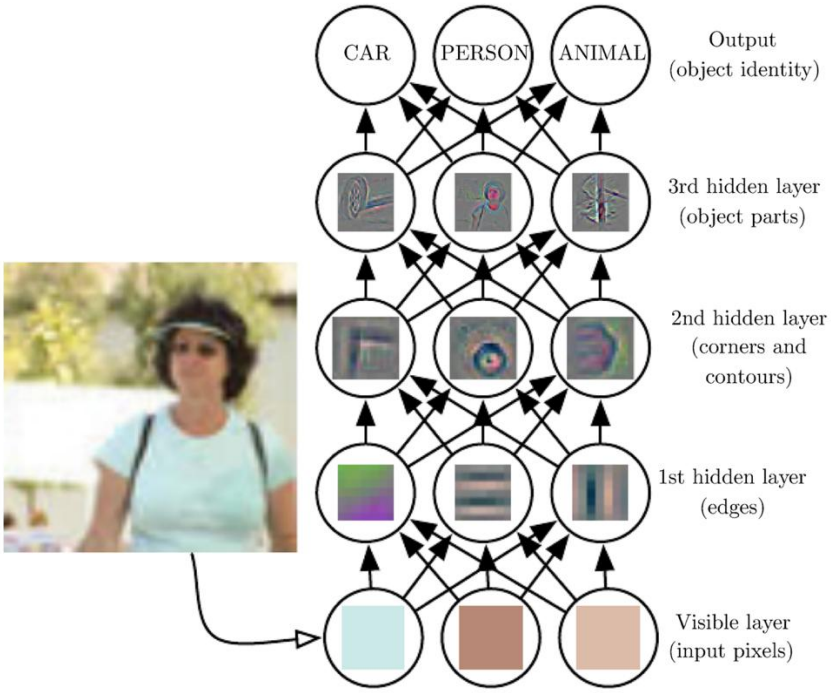
La arquitectura de una CNN se fundamenta en principios extraídos del estudio de la corteza visual de los mamíferos, y su poder reside en tres ideas clave: interacciones dispersas, parámetros compartidos y representaciones equivariantes (Goodfellow et al., 2016). En primer lugar, las CNN se distinguen por implementar una conectividad local o interacciones dispersas. A diferencia de las redes densas tradicionales donde existe una conexión total entre capas, en este modelo cada neurona procesa información únicamente de una subregión específica de la entrada, denominada campo receptivo. Tal arquitectura es fundamentada en el funcionamiento del sistema visual biológico y faculta a la red para especializarse en la detección inicial de patrones elementales como bordes y vértices (Berzal, 2018).

En segundo lugar, las CNN emplean el concepto de parámetros compartidos (parameter sharing). Esto significa que un mismo conjunto de pesos, que conforman una máscara o filtro (kernel), se aplica de manera iterativa a lo largo de toda la imagen de entrada. Al deslizar el mismo filtro por toda la imagen, la red puede detectar la misma característica (por ejemplo, una línea vertical) sin importar en qué posición de la imagen aparezca. Esta propiedad no solo reduce drásticamente el número de parámetros a aprender, facilitando el entrenamiento, sino que también introduce la equivarianza a la traslación, done "si el objeto en la entrada se mueve, su representación se moverá la misma cantidad en la salida” (Goodfellow et al., 2016, p. 338).

Según Berzal (2018), la estructura estándar de estas redes integra secuencialmente tres categorías de capas: primero, las etapas convolucionales responsables de la extracción de rasgos visuales; segundo, las capas de submuestreo (pooling) que disminuyen la resolución espacial para optimizar el cómputo y otorgar invarianza frente a desplazamientos leves; y finalmente, las capas densas o totalmente conectadas, encargadas de interpretar las características abstractas resultantes para ejecutar la clasificación definitiva.

Figura 2

Jerarquía de características en una red neuronal convolucional



Nota. La figura ilustra el modelo conceptual de cómo una red de aprendizaje profundo procesa una imagen: las primeras capas identifican patrones simples como bordes; capas posteriores combinan estos para reconocer formas complejas como contornos, esquinas y, finalmente, partes de objetos, como un ojo o una nariz. La capa de salida utiliza estas representaciones de alto nivel para realizar la clasificación final. Tomado de *Deep Learning* (p. 6), por I. Goodfellow, Y. Bengio, y A. Courville, 2016.

6.2.6 Aprendizaje por Transferencia (Transfer Learning). El Aprendizaje por Transferencia es entendida la situación en la que el conocimiento aprendido en un entorno o

distribución de datos (P_1) es utilizado para mejorar la capacidad de generalización en un segundo entorno (P_2) (Goodfellow et al., 2016). La premisa fundamental de esta técnica es que, aunque el aprendiz deba ejecutar dos o más tareas diferentes, muchos de los factores que explican las variaciones en la distribución de datos original (P_1) son también relevantes para capturar las variaciones en la nueva tarea.

Esta técnica es particularmente efectiva en el aprendizaje supervisado. Por ejemplo, se puede aprender a clasificar un conjunto de categorías visuales (como perros y gatos) en un primer entorno, para luego aplicar ese conocimiento al aprendizaje de un conjunto diferente de categorías (como hormigas y avispas) en un segundo entorno. Goodfellow et al. (2016) señalan que, si el primer entorno cuenta con una cantidad de datos significativamente mayor, esto "puede ayudar a aprender representaciones que son útiles para generalizar rápidamente a partir de muy pocos ejemplos extraídos de P_2" (p. 536). El conocimiento transferido consiste en nociones de bajo nivel compartidas entre múltiples categorías visuales, como la detección de bordes, formas, y los efectos de cambios geométricos o de iluminación.

6.2.7 Taxonomía vegetal. La taxonomía vegetal es la ciencia de la clasificación de organismos, organizada en un sistema jerárquico que agrupa a las plantas según similitudes morfológicas y filogenéticas (Michaels et al., 2022), y que organiza la vida vegetal en rangos anidados de creciente especificidad, siendo los más relevantes para este estudio: a) la Familia, que agrupa géneros con características estructurales y reproductivas comunes (por ejemplo, la familia *Fabaceae* agrupa a todas las plantas leguminosas); b) el Género, que reúne a especies estrechamente emparentadas dentro de una familia (ej., el género *Trifolium* agrupa a las diferentes especies de tréboles dentro de las *Fabaceae*); y c) la Especie, que constituye la unidad fundamental de clasificación (p. ej., *Trifolium repens* designa específicamente al trébol blanco).

Para este proyecto, el nivel taxonómico objetivo del modelo de clasificación es la especie, ya que es el de mayor especificidad y valor operativo para aplicaciones de monitoreo de biodiversidad. La identificación a nivel de especie se corresponde con la nomenclatura binomial de Linneo, donde cada planta es unívocamente designada por su Género y epíteto específico (por ejemplo *Trifolium repens* designa específicamente al trébol blanco) (Michaels et al., 2022). Por lo tanto, cada clase de salida del modelo corresponderá a una especie distinta de la flora bogotana.

6.3 Marco ingenieril

6.3.1 Arquitecturas de redes neuronales convolucionales. Luego de la revisión de los antecedentes técnicos en reconocimiento de flora por visión computacional (apartado 6.1.4) y teniendo en cuenta los requerimientos de este proyecto inherentes a la latencia de inferencia, el consumo de memoria y el gasto energético propios de una implementación en dispositivos móviles, se han seleccionado y analizado cuatro arquitecturas que representan el estado del arte en diferentes puntos de este espectro: ResNet-50, MobileNetV2, MobileNetV3 y EfficientNet-B0. A continuación, se describen los principios de diseño, optimización y consideraciones de cada una.

6.3.1.1 ResNet. Esta arquitectura se fundamenta en redes residuales; antes de su desarrollo se observaba que añadir simplemente más capas a las redes neuronales profundas resultaba en un problema de degradación del rendimiento; redes más profundas llegaban a tener un error de entrenamiento y de prueba superior al de sus contrapartes más superficiales. Para solventar esta dificultad, He et al. (2015) introducen un marco de aprendizaje residual, teniendo como hipótesis que, en lugar de esperar que un conjunto de capas apiladas aprenda directamente una función subyacente $\mathcal{H}(x)$, es más sencillo que aprendan una función residual $\mathcal{F}(x) := \mathcal{H}(x) - x$. La función original se reformula entonces como $\mathcal{F}(x) + x$. (p. 3), materializada en la arquitectura a través de los denominados bloques residuales, cuya innovación clave es la conexión de atajo (shortcut connection), la cual toma la entrada x de un bloque y la suma, elemento a elemento, a la salida de las capas convolucionales de dicho bloque, $\mathcal{F}(x)$. La operación se define mediante la siguiente expresión:

$$y = \mathcal{F}(x, \{W_i\}) + x$$

donde x e y son los vectores de entrada y salida del bloque, y $\mathcal{F}(x, \{W_i\})$ representa el mapeo residual que las capas aprenden. La hipótesis subyacente es que resulta más sencillo para un conjunto de capas optimizar un mapeo residual hacia cero —lo que equivale a aprender una transformación de identidad— que ajustar los pesos para que emulen dicha transformación desde cero.

ResNet posee diferentes variantes (ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152) las cuales se distinguen principalmente por el número total de capas que componen la red, lo cual se logra apilando un número variable de bloques residuales. Las versiones más

profundas como ResNet-50 y superiores introducen, además, una optimización llamada "bloque de cuello de botella" *(bottleneck block)*, que utiliza convoluciones de 1x1 para reducir y luego restaurar la dimensionalidad de los canales, haciendo que los bloques profundos sean computacionalmente más eficientes (He et al., 2015).

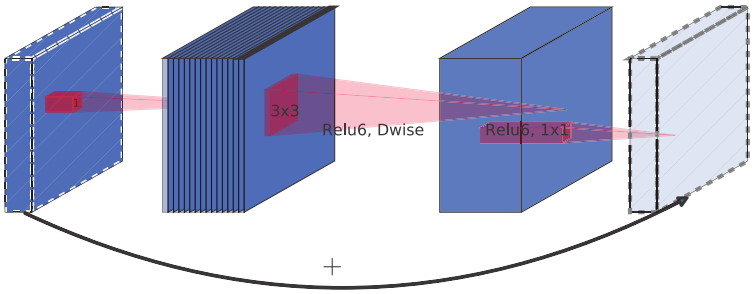
6.3.1.2 MobileNetV2. Esta arquitectura presenta una mejora significativa en la eficiencia de las redes convolucionales a través de dos innovaciones arquitectónicas principales. La primera es el uso de convoluciones separables en profundidad *(depthwise separable convolutions)*, que factorizan una convolución estándar en dos pasos: una convolución ligera que filtra espacialmente cada canal de entrada de forma independiente *(depthwise)*, y una convolución de 1x1 que combina linealmente los canales resultantes *(pointwise)* (Sandler et al., 2019).

La segunda y más importante innovación es el bloque de residuos invertidos *(inverted residual block)*, donde, a diferencia de los bloques de ResNet que tienen una estructura "ancho-estrecho-ancho" (donde los canales se comprimen, se procesan y luego se expanden), MobileNetV2 invierte esta lógica: el bloque comienza con una capa de cuello de botella *(bottleneck)* con pocos canales, la expande a una dimensión mucho mayor con una convolución de 1x1, aplica la convolución depthwise de 3x3 en este espacio expandido, y finalmente la proyecta de nuevo a una representación de cuello de botella con una convolución lineal de 1x1. Las conexiones de atajo, análogas a las de ResNet, se establecen entre los cuellos de botella de baja dimensión. La estructura invertida es significativamente más eficiente en términos de memoria, ya que las conexiones de atajo enlazan las representaciones más compactas.

Además, Sandler et al. (2019) argumentan que el uso de una función de activación no lineal –ReLU por ejemplo– en los cuellos de botella puede destruir información importante (p. 3). Por ello, la capa de proyección final de cada bloque es lineal, es decir, no tiene función de activación. Esta característica, denominada cuello de botella lineal, es clave para preservar la capacidad representativa del modelo.

Figura 3

Bloque de construcción de MobileNetV2 con residuo invertido



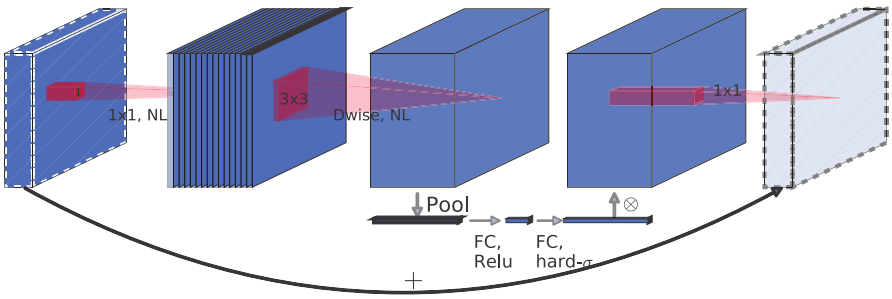
Nota. El diagrama muestra el bloque fundamental de la arquitectura MobileNetV2: una capa inicial de 1x1 expande los canales, una convolución depthwise de 3x3 aplica el filtrado espacial, y una capa de proyección lineal de 1x1 reduce nuevamente los canales. Tomado de *MobileNetV2: Inverted Residuals and Linear Bottlenecks* (p. 3), por M. Sandler et al., 2019.

6.3.1.3 MobileNetV3. Esta arquitectura propuesta presentada por Howard et al. (2019) no es una reinvención completa, sino una optimización refinada de MobileNetV2. Su desarrollo combina la búsqueda de arquitectura de red neuronal (Network Architecture Search - NAS) con nuevas mejoras a nivel de bloque. En primer lugar, la macro y microestructura de la red no fue diseñada manualmente, sino descubierta por un algoritmo de búsqueda que optimiza el balance entre precisión y latencia para CPUs de teléfonos móviles específicos.

En segundo lugar, se introdujeron varias mejoras arquitectónicas. Una de las más significativas es la incorporación de bloques Squeeze-and-Excite (SE), un mecanismo de atención ligero que permite a la red realizar una recalibración de características canal por canal, dando más peso a las características más informativas. En la Figura 4 se ilustra tal incorporación.

Figura 4

Bloque de construcción de MobileNetV3



Nota. El diagrama ilustra el bloque optimizado de la arquitectura MobileNetV3. Hereda la estructura de residuo invertido de MobileNetV2, pero incorpora el módulo de Squeeze-and-Excite (representado por el bloque "Pool" y las capas "FC"). Tomado de *Searching for MobileNetV3* (p. 3), por A. Howard et al., 2019.

Otra innovación es la introducción de una nueva función de activación no lineal llamada h-swish, versión modificada y computacionalmente más barata de la función swish, la cual, a pesar de sus beneficios en precisión, presenta un costo de latencia debido a la complejidad de la función sigmoide en su cálculo:

$$\text{swish } x \;=\; x \cdot \sigma(x)$$

Para mitigar este costo, MobileNetV3 propone la función h-swish, que aproxima la función swish, pero reemplazando la función sigmoide por su análogo lineal por tramos ReLU6, lo que reduce significativamente la latencia sin una pérdida considerable de precisión y con tolerancia a la cuantización:

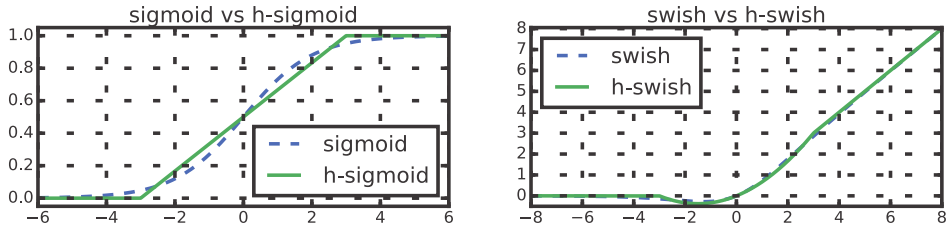
$$h - swish[x] \;=\; x\frac{ReLU6(x+3)}{6}$$

La

Figura 5 ilustra la comparativa entre ambas funciones, evidenciando cómo la versión *hard* mantiene una forma similar a la original con una implementación computacionalmente más simple.

Figura 5

Comparativa de funciones de activación soft y hard



Nota. La figura ilustra la función sigmoide y su aproximación eficiente h-sigmoid (izquierda), así como la función swish y su contraparte h-swish (derecha). Tomado de *Searching for MobileNetV3* (p. 5), por A. Howard et al., 2019.

Finalmente, MobileNetV3 también rediseña las capas finales de la red para reducir la latencia en la etapa de generación de características, eliminando capas computacionalmente costosas sin una pérdida significativa de precisión.

6.3.1.4 EfficientNet. Las arquitecturas vistas anteriormente se enfocan en modificar el diseño de los bloques de construcción para mejorar la precisión o la eficiencia. En contraste, la familia de arquitecturas EfficientNet aborda el problema desde una perspectiva diferente: en lugar de proponer un nuevo bloque, investiga cómo escalar de manera óptima una arquitectura base para obtener el mejor balance entre precisión y recursos computacionales (Tan & Le, 2020). Históricamente, las redes se escalaban aumentando una de tres dimensiones: la profundidad añadiendo más capas, la anchura aumentando el número de canales en cada capa, o la resolución de la imagen de entrada. Sin embargo, Tan & Le (2020) demostraron empíricamente que escalar una sola de estas dimensiones produce rendimientos decrecientes.

Para solucionar esta limitación, propusieron un método de escalado compuesto (compound scaling). La intuición es que las tres dimensiones de escalado no son independientes: imágenes de mayor resolución requieren redes más profundas para tener un campo receptivo más grande que capture patrones globales, y a su vez, redes más anchas para capturar patrones más finos en la imagen de alta resolución. El método de escalado compuesto formaliza esta idea al escalar las tres

dimensiones de manera uniforme utilizando un coeficiente compuesto ϕ, que es un valor especificado por el usuario para controlar el tamaño del modelo. La profundidad (d), anchura (w) y resolución (r) se escalan según las siguientes reglas:

$$d = \alpha^{\phi}$$

$$w = \beta^{\phi}$$

$$r = \gamma^{\phi}$$

donde α, β, y γ son constantes determinadas mediante una búsqueda en una pequeña malla de valores sobre un modelo base inicial, bajo la restricción de que $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.

Para aplicar este principio, los autores primero diseñaron una nueva arquitectura base altamente eficiente, denominada EfficientNet-B0, utilizando una búsqueda de arquitectura neuronal (NAS) y que emplea bloques de residuo invertido similares a los de MobileNetV2. Luego, aplicaron el método de escalado compuesto sobre este modelo base para obtener una familia de redes (EfficientNet-B1 a B7) que superaron el estado del arte. El resultado es una arquitectura que, para un presupuesto computacional dado medido en FLOPS, alcanza una precisión significativamente mayor que las arquitecturas anteriores.

6.3.2 Estrategias de Transfer Learning. Después de haber definido el aprendizaje por transferencia (Sección 6.2.6) se detallarán las estrategias que se pretenden usar. Dentro de la literatura se describen principalmente dos abordajes básicos para reutilizar un modelo pre-entrenado: emplearlo como un extractor de características estático o someterlo a un proceso de ajuste fino *(fine-tuning)* (Berzal, 2018). También se describe un abordaje avanzado, denominado métodos de ensamble.

El primer abordaje, y el más simple computacionalmente se usa bastante cuando el dataset de destino es pequeño y muy diferente del dataset original (ImageNet), ya que evita el sobreajuste que podría ocurrir al intentar re-entrenar la red completa. Consiste en utilizar la base convolucional de una red pre-entrenada como un extractor de características de propósito general, donde se congela el entrenamiento de todas las capas convolucionales del modelo (es decir, sus pesos no se actualizan) y se utiliza para procesar las imágenes del nuevo dataset. La salida de la última capa

convolucional, conocida como mapa de características, se convierte en un vector que representa la imagen en un espacio de características de alto nivel y que se utiliza como entrada para entrenar un clasificador nuevo y mucho más pequeño (usualmente una o dos capas completamente conectadas) que aprenderá a mapear estas características extraídas a las clases específicas del problema (Berzal, 2018), para este proyecto sería la flora bogotana.

El segundo abordaje, más potente y computacionalmente más intensivo, es el ajuste fino o fine-tuning. Esta estrategia entrena un nuevo clasificador y también descongela algunas de las capas superiores de la base convolucional pre-entrenada y continúa su entrenamiento con el nuevo dataset (Berzal, 2018). El fundamento detrás de este método es el mismo por el que funciona una CNN: sus primeras capas aprenden características muy genéricas (bordes, colores, texturas), mientras que las capas más profundas aprenden características más complejas y específicas del dataset original (patrones de ojos, ruedas, etc.). Al realizar el ajuste fino, se busca especializar estas características de alto nivel a los matices del nuevo dominio.

La decisión sobre cuántas capas descongelar es un hiperparámetro muy sensible que depende del tamaño y la similitud del nuevo dataset. Para datasets grandes y similares a ImageNet, se puede optar por un ajuste fino de toda la red. Para datasets más pequeños pero similares, es común congelar las primeras capas y ajustar solo las últimas. De igual manera, este proceso debe realizarse con una tasa de aprendizaje muy baja para evitar que las actualizaciones de los gradientes, inicialmente muy grandes, destruyan las representaciones de características ya aprendidas (Berzal, 2018). Investigaciones recientes han explorado estrategias avanzadas como el descongelamiento progresivo (progressive unfreezing), que descongela las capas gradualmente para lograr una adaptación más estable, reportando mejoras en la precisión (Picek et al., 2022).

De manera más avanzada, los métodos de ensamble (ensemble methods), en lugar de optimizar un único modelo combinan las predicciones de múltiples modelos para obtener un resultado final más preciso, y se basan en el principio que diferentes modelos, entrenados con distintas inicializaciones o sobre subconjuntos de datos ligeramente diferentes, cometerán errores distintos, y al promediar o combinar sus predicciones, estos errores no correlacionados tienden a anularse, resultando en una clasificación final más estable y precisa que la que podría ofrecer cualquiera de los modelos individuales por sí solo (Pérez Esteve, 2020).

Se encontraron estudios que han alcanzado resultados sobresalientes mediante la combinación de arquitecturas heterogéneas y validan este método como estrategia de refinamiento. Por ejemplo, un modelo de ensamble denominado Plant-CNN-ViT, que integra las predicciones de Vision Transformer, ResNet-50, DenseNet-201 y Xception, ha reportado un 100% de precisión en múltiples datasets de referencia (Kaur et al., 2023). De manera similar, un enfoque que combina ResNet-9, VGG-19 y DenseNet-121 logró una precisión del 97.56% en la clasificación de enfermedades del arroz (Picek et al., 2022).

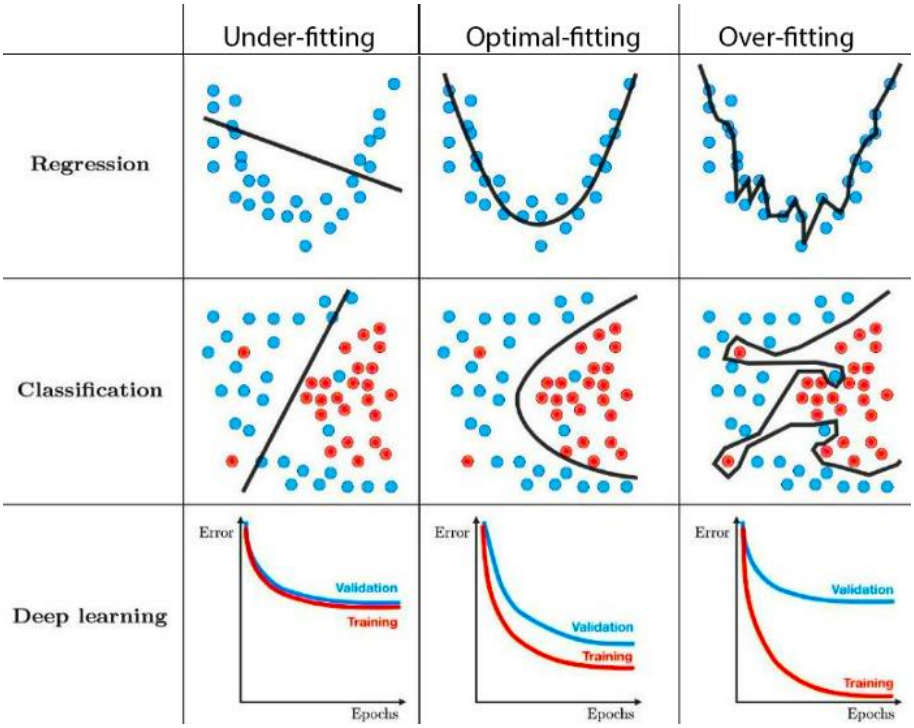
La principal desventaja de los métodos de ensamble es el incremento en el costo computacional, tanto en la fase de entrenamiento como en la de inferencia al requerir entrenar, almacenar y ejecutar múltiples redes neuronales en lugar de una sola, multiplicando los requerimientos de memoria, tiempo de procesamiento y consumo energético. Por esta razón, aunque los métodos de ensamble representan el límite superior de la precisión alcanzable, su implementación se considera inviable para el alcance de este proyecto, que prioriza la eficiencia y la viabilidad de ejecución en un dispositivo móvil con recursos limitados.

6.3.3 Regularización y prevención del sobreaprendizaje. Un requisito fundamental en el aprendizaje automático es que el desempeño del sistema no se limite a los datos de entrenamiento, sino que se extienda a información nueva, propiedad conocida como generalización. Según (Goodfellow et al., 2016), para alcanzar este objetivo se emplean técnicas de regularización, las cuales se definen como modificaciones algorítmicas orientadas a minimizar el error de generalización, diferenciándose de aquellas que solo buscan reducir el error de entrenamiento (p. 228). En esencia, son un conjunto de estrategias diseñadas para, entre otras cosas, prevenir el sobreaprendizaje y mejorar la capacidad de inferencia del modelo.

El sobreaprendizaje (overfitting en inglés) es un fenómeno que ocurre cuando un modelo se acopla excesivamente al conjunto de entrenamiento, sacrificando su capacidad de inferencia en datos nuevos. Según Goodfellow et al. (2016), este fenómeno se diagnostica matemáticamente cuando existe una divergencia sustancial entre un error de entrenamiento muy bajo y un error de prueba elevado. En la práctica, esto implica que el algoritmo, en lugar de abstraer las características estructurales de los datos, ha codificado el ruido estocástico y los sesgos del muestreo (Berzal, 2018), resultando en un sistema que aparenta alta precisión, pero su rendimiento se degrada notablemente al enfrentarse a datos del mundo real, volviéndose inútil para la aplicación práctica.

Figura 6

Comparativa de subajuste, ajuste óptimo y sobreajuste



Nota. La figura ilustra los tres escenarios de ajuste del modelo en tareas de regresión, clasificación y aprendizaje profundo. El sobreajuste (derecha) ocurre cuando el modelo se adapta excesivamente al ruido de los datos de entrenamiento, perdiendo su capacidad de generalización. Tomado de *Techniques for handling underfitting and overfitting in Machine Learning*, por M. S. Minhas, 2021, Towards Data Science.

La causa fundamental del sobreaprendizaje es una capacidad del modelo excesivamente alta en relación con la complejidad y cantidad de los datos de entrenamiento (Berzal, 2018), por lo que, si la capacidad de un modelo es demasiado grande, puede encontrar una infinidad de soluciones complejas que se ajustan perfectamente a los datos de entrenamiento, incluyendo su ruido, en lugar de converger hacia la solución más simple y generalizable. Como lo expresa el aforismo del estadístico George Box: "todos los modelos están equivocados, pero algunos son útiles" (Box, 1979, citado en Berzal, 2018, p. 428). El objetivo de las técnicas de regularización es, precisamente, guiar el proceso de entrenamiento para que el modelo resultante sea uno de los

"útiles", penalizando la complejidad innecesaria y fomentando la generalización. A continuación, se presentan técnicas de regularización aplicables a este proyecto, estas son: El dropout, el early stopping y el data augmentation.

6.3.3.1 Dropout. Esta técnica de regularización está diseñada para prevenir la coadaptación de las neuronas en una red (Srivastava et al., 2014, citado en Berzal, 2018). Durante cada iteración del entrenamiento, dropout apaga de forma aleatoria y temporal la salida de un subconjunto de neuronas en las capas ocultas (es decir, las multiplica por cero). Típicamente, cada neurona tiene una probabilidad p de ser abandonada en una iteración dada, donde un valor común para p es 0.5 (Goodfellow et al., 2016).

Este proceso tiene dos interpretaciones complementarias. Por un lado, puede ser visto como un método para entrenar de manera eficiente un gran ensamble de redes neuronales con arquitecturas diferentes, donde cada iteración de entrenamiento se aplica sobre una "sub-red" distinta, y el hecho de que todas estas sub-redes compartan parámetros funciona como un mecanismo de regularización muy potente (Berzal, 2018). Por otro lado, al anular neuronas aleatoriamente, se evita que las neuronas dependan excesivamente de la presencia de otras neuronas específicas para corregir sus errores, obligándolas a aprender características independientes, que sean útiles en una variedad de contextos internos, previniendo así las "complejas co-adaptaciones" que a menudo conducen al sobreaprendizaje (Hinton et al., 2012, citado en Berzal, 2018, p. 462). Durante la fase de inferencia o prueba, no se aplica dropout y se utilizan todas las neuronas, pero sus salidas se escalan para compensar el hecho de que más unidades están activas que durante el entrenamiento.

6.3.3.2 Parada temprana (early stopping). La parada temprana se define como una estrategia de regularización temporal que supervisa métricas en un conjunto de validación aislado, el cual no interviene en la actualización de los parámetros de la red. Según Goodfellow et al. (2016), mientras que el error de entrenamiento tiende a descender de manera continua (monótona), el error de validación describe una trayectoria en forma de 'U'; el punto de inflexión donde dicho error deja de disminuir y comienza a ascender constituye el indicador técnico del inicio del sobreajuste.

La estrategia de parada temprana consiste en detener el proceso de entrenamiento en el momento en que el error en el conjunto de validación deja de mejorar o empieza a incrementarse. Específicamente, la implementación técnica consiste en monitorear el desempeño sobre el conjunto de validación, persistiendo la configuración de la red (parámetros) únicamente cuando se supera el mejor resultado histórico. El ciclo de entrenamiento se aborta si transcurre un intervalo de épocas predefinido —conocido como paciencia— sin que se reduzca el error. Según Berzal (2018), esta estrategia opera como un regularizador efectivo al impedir que el modelo continúe su convergencia hacia soluciones que se ajustan excesivamente a los datos de entrenamiento en detrimento de la validación.

6.3.3.3 Data Augmentation. Esta técnica consiste en ampliar artificialmente el conjunto de datos de entrenamiento mediante la generación de nuevas muestras a partir de las existentes, aplicando transformaciones que preservan la etiqueta de la clase (Goodfellow et al., 2016). Para tareas de clasificación de imágenes, el objetivo principal es lograr que el modelo sea invariante a una amplia variedad de transformaciones que no alteran la identidad semántica del objeto.

La implementación de esta técnica implica aplicar un conjunto de operaciones sobre las imágenes de entrenamiento: pueden ser geométricas, como traslaciones, rotaciones, cambios de escala, recortes o volteos; o bien pueden ser transformaciones fotométricas, como ajustes en el brillo, contraste y saturación del color (Berzal, 2018; Goodfellow et al., 2016). Al entrenar el modelo con estas versiones modificadas, se le fuerza a aprender las características esenciales del objeto en lugar de memorizar la disposición específica de los píxeles de las imágenes originales, preparando a la red ante las variaciones que encontrará en datos del mundo real.

Figura 7

Ejemplo de transformación de data augmentation

Nota. La figura muestra una imagen original del conjunto de datos y dos ejemplos de imágenes aumentadas generadas a partir de ella. Se aplicaron transformaciones de rotación, translación y deformación de manera aleatoria.

6.3.4 Métricas de evaluación del modelo. La validación del sistema de aprendizaje automático exige una cuantificación minuciosa de su desempeño. Para este fin, se seleccionaron indicadores estandarizados en el dominio de visión por computador, los cuales permiten auditar la capacidad predictiva del clasificador y detectar patrones de error. El análisis se sustenta en el monitoreo de la función de pérdida y, fundamentalmente, en la construcción de la matriz de confusión.

Dicha matriz constituye el instrumento principal para contrastar las inferencias del modelo frente a las etiquetas verdaderas. Se define como una estructura tabular cuadrada de dimensiones K x K (donde K es el número de categorías). Siguiendo la convención descrita por Berzal (2018), las filas denotan las clases reales, mientras que las columnas representan las predicciones del sistema. Matemáticamente, cada celda (i, j) cuantifica las instancias de la clase i que fueron asignadas a la clase j. Por consiguiente, la traza de la matriz (diagonal principal) agrupa las predicciones acertadas, en tanto que los valores fuera de esta diagonal reflejan las clasificaciones erróneas. De esta distribución se extraen los cuatro estadísticos base por clase: Verdaderos Positivos (TP), Falsos Positivos (FP), Falsos Negativos (FN) y Verdaderos Negativos (TN).

La exactitud es la métrica más intuitiva y comúnmente utilizada. Mide la proporción de predicciones correctas sobre el total de predicciones realizadas. Aunque es una medida útil para tener una visión general del rendimiento, puede ser engañosa en problemas con un fuerte desbalance de clases, ya que un modelo podría alcanzar una alta exactitud simplemente prediciendo siempre la clase mayoritaria (Berzal, 2018). Se calcula de la siguiente manera:

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{\text{Aciertos}}{\text{Total de muestras}}$$

Para una evaluación más granular, especialmente en contextos de clases desbalanceadas, se recurre a la precisión y la sensibilidad. La precisión mide la proporción de predicciones positivas que fueron realmente correctas. Es una métrica de calidad, que responde a la pregunta: de todas las veces que el modelo predijo una clase, ¿cuántas veces acertó? (Berzal, 2018).

$$\text{Precisión} = \frac{TP}{TP + FP}$$

La sensibilidad (recall), también conocida como tasa de verdaderos positivos (TPR) o exhaustividad, mide la proporción de positivos reales que fueron correctamente identificados por el modelo. Es una métrica de cantidad, que responde a la pregunta: de todas las muestras que realmente pertenecían a una clase, ¿cuántas fue capaz de encontrar el modelo? (Berzal, 2018).

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

La Puntuación F1 es la media armónica de la precisión y la sensibilidad. Se utiliza como una métrica única que resume el equilibrio entre ambas, ya que penaliza fuertemente los valores extremos. Un modelo solo obtendrá una Puntuación F1 alta si tanto su precisión como su sensibilidad son altas (Berzal, 2018).

$$\text{F1-Score} = 2 \cdot \frac{\text{Precisión} \cdot \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}}$$

Para problemas multi-clase como la clasificación de flora, estas métricas se calculan para cada clase individualmente y luego se promedian. Se emplean dos estrategias de promediado: *Macro Average*, que calcula la media aritmética de las métricas de todas las clases, tratando a cada clase por igual; y *Weighted Average*, que pondera la media según el número de muestras de cada clase (su soporte), dando más importancia a las clases con más instancias.

6.3.5 Frameworks y tecnologías de soporte para Machine Learning. Para poder llevar a cabo con éxito el ciclo de vida de la construcción del modelo, se eligieron una serie de tecnologías adecuadas para la tarea.

Se seleccionó Python como lenguaje de programación debido a su consolidación como estándar de facto abierto en la comunidad de ciencia de datos e inteligencia artificial. Reside en un ecosistema de bibliotecas especializadas de código abierto que simplifican las tareas inherentes al campo, desde la manipulación de datos hasta la materialización de los modelos de aprendizaje. Para gestionar este ecosistema y garantizar la reproducibilidad, se utilizó Conda como gestor de paquetes y entornos para la crear de ambientes de software aislados con versiones de bibliotecas específicas y evitar así conflictos de dependencias. La experimentación y documentación del proceso se llevaron a cabo en Jupyter Notebooks, un entorno de computación interactiva que facilita la combinación de código ejecutable al permitir dividir los pasos en bloques ejecutables de manera secuencial con posibilidad de añadir anotaciones y ver salidas gráficas y de texto por cada bloque para el prototipado rápido y el análisis exploratorio.

Sobre esta base de desarrollo se implementaron los frameworks especializados en aprendizaje profundo, operando en distintos niveles de abstracción, desde la computación numérica de bajo nivel hasta la definición de alto nivel de las arquitecturas de red.

6.3.5.1 TensorFlow. Es una biblioteca de código abierto que permite definir y ejecutar cálculos que involucran tensores en grafos computacionales (Martín Abadi et al., 2015) . Su capacidad para ejecutarse de manera eficiente en diferentes plataformas de hardware, como CPUs y GPUs, es la base de alto rendimiento sobre la cual se construyó y entrenó el modelo.

6.3.5.2 Keras. Interfaz de alto nivel sobre TensorFlow que una API modular, abstrae y simplifica el proceso de construcción de modelos de aprendizaje profundo al proporcionar componentes preconfigurados como capas, funciones de activación, optimizadores y funciones de entrenamiento (Chollet & others, 2015).

6.3.5.3 TensorFlow Lite. Su función es optimizar los modelos de TensorFlow ya entrenados para su ejecución en dispositivos móviles y sistemas embebidos. Ofrece herramientas para optimización de modelos mediante varias modalidades, así como los intérpretes para las diferentes arquitecturas, sistemas y entornos de ejecución móviles y de bajo consumo.

El ciclo de vida del modelo de aprendizaje automático es soportado, a su vez, por un conjunto de bibliotecas especializadas que facilitan tareas críticas como la manipulación de datos, la evaluación del rendimiento y la monitorización del entrenamiento.

6.3.5.4 Scikit-learn. Es una biblioteca de aprendizaje automático en Python que proporciona herramientas para tareas como la clasificación, regresión y clustering. Se usó para la división de conjuntos de datos y para el cálculo de métricas de evaluación de rendimiento de modelos como complemento a Keras y TensorFlow.

6.3.5.5 NumPy y Pandas. NumPy es la biblioteca fundamental para la computación científica en Python con soporte para tensores y un conjunto de funciones matemáticas de alto nivel para operar sobre ellos. Pandas se construye sobre NumPy y provee estructuras de datos y herramientas de análisis para manipular datos tabulares y series temporales.

6.3.5.6 TensorBoard. Es un kit de herramientas de visualización integrado en el ecosistema de TensorFlow. Permite monitorizar y depurar el proceso de entrenamiento de un modelo, provee un sistema de guardado de logs e integración con Keras para registrar métricas en tiempo real, así como la estructura del grafo computacional y los perfiles de rendimiento. También se usa con fines de revisión histórica de los procesos de entrenamiento, al poderse ejecutar de manera portable junto con los logs pertinentes.

Finalmente, se usaron sistemas de control de versiones para garantizar la reproducibilidad y trazabilidad de los experimentos mediante herramientas de control de versiones.

6.3.5.7 Git y GitHub. Git es un sistema de control de versiones distribuido, diseñado para gestionar el código fuente de manera eficiente. GitHub es una plataforma de alojamiento para repositorios de Git que facilita el trabajo colaborativo y la gestión de proyectos de software.

6.3.5.8 DVC (Data Version Control). Es un sistema de control de versiones de código abierto diseñado para el ciclo de vida del aprendizaje automático para gestionar artefactos de gran tamaño, como datasets y modelos sin almacenarlos directamente en los repositorios de Git (ya que este último no gestiona bien los binarios y archivos grandes), permitiendo la trazabilidad y reproducibilidad de las fases experimentales (The DVC team and contributors, 2025)

6.3.6 Frameworks y tecnologías de soporte para aplicación móvil. Para dar respuesta a los alcances definidos, se seleccionó un stack tecnológico que permitiera desarrollar la aplicación multiplataforma usando solamente una base de código. A continuación, se detallan las tecnologías utilizadas.

6.3.6.1 React Native. Es un framework de código abierto desarrollado por Meta que permite construir aplicaciones móviles nativas utilizando el paradigma de la librería React, escrita a su vez en el lenguaje TypeScript. Su principal ventaja reside en su arquitectura "aprender una vez, escribir en cualquier lugar" (learn once, write anywhere), que permite a los desarrolladores utilizar sus conocimientos de desarrollo web para crear interfaces de usuario móviles (Meta Platforms, 2025). La arquitectura de React Native está diseñada para que la lógica de la aplicación, escrita en JavaScript, se ejecute de manera desacoplada del hilo principal de la interfaz de usuario nativa para que haya un rendimiento fluido semejante al código nativo. Su funcionamiento se ilustra de mejor manera en la Figura 8 y se describe a continuación:

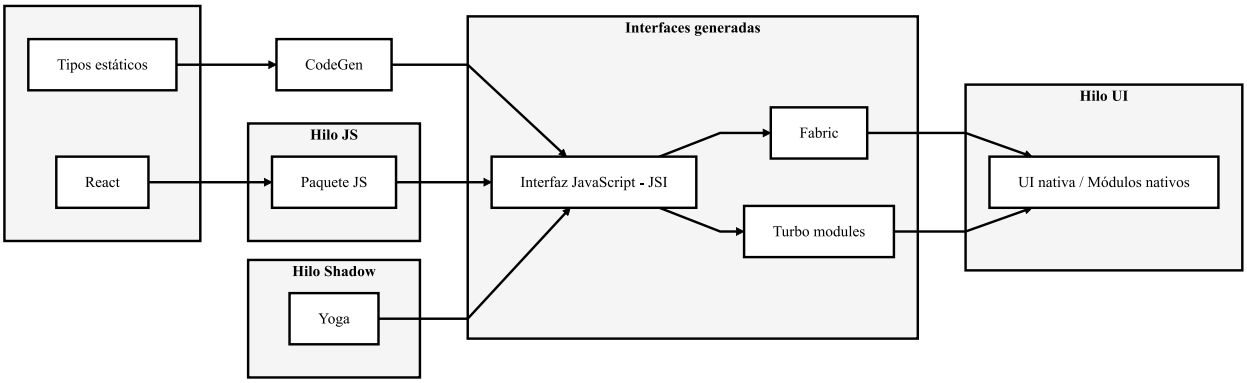
En el centro del funcionamiento de este framework se encuentra la JavaScript Interface (JSI), una capa de comunicación en C++ que conecta el motor de JavaScript con los hilos nativos. A diferencia de sistemas basados en paso de mensajes, la JSI permite mantener referencias directas entre objetos de ambos entornos, lo que habilita la ejecución síncrona y la concurrencia en operaciones críticas.

Sobre esta base se organizan dos componentes principales. Por un lado, Codegen genera de manera automática las interfaces necesarias para enlazar los tipos definidos en TypeScript con el entorno nativo. Dichas interfaces son utilizadas por los Turbo Modules, la implementación moderna de los módulos nativos, que se cargan únicamente cuando son requeridos para optimizar el uso de recursos. Por otro lado, la renderización de la interfaz gráfica se gestiona mediante Fabric, el cual se comunica directamente con el hilo principal a través de la JSI, garantizando fluidez en respuesta ante las interacciones del usuario. El cálculo de disposición de los elementos se realiza con Yoga, un motor que implementa el modelo de diseño Flexbox que se utiliza en la web y permite organizar la disposición de los elementos de una interfaz por filas y columnas flexibles ante los diferentes tamaños de pantalla.

El proceso de renderizado se organiza en tres fases: 1) Render, donde JavaScript construye un árbol de representación de la interfaz (Shadow tree); 2) Commit, en el que Yoga calcula posiciones y dimensiones; y 3) Mount, donde el árbol se materializa en vistas nativas (Traducidos a UIView en iOS con Swift o android.view en Android con Kotlin) que se muestran en pantalla.

Figura 8

Arquitectura de React Native



Nota. La JSI conecta el entorno de JavaScript con los componentes nativos. Codegen y los Turbo Modules facilitan la comunicación, mientras que Fabric y Yoga gestionan el renderizado y la disposición de la interfaz.

6.3.6.2 Expo. Es un conjunto de herramientas y un framework de código abierto construido sobre React Native que abstrae la gestión del flujo de trabajo, compilación y ensamble del desarrollo móvil. Con su uso se elimina la necesidad de configurar directamente los entornos de compilación nativos, y se tiene acceso a funcionalidades preconstruidas para acceder al hardware y funcionalidades nativas del dispositivo, como lo es la cámara, la gestión de permisos, base de datos local, mapas, etc. De igual manera proporciona un estilo de navegación basada en carpetas para definir las rutas de la aplicación mediante la estructura de los archivos del proyecto. Otra característica es el soporte para compilación en la nube y distribución de actualizaciones en caso de publicarse en tienda de aplicaciones (Expo, 2025).

6.3.6.3 Ejecución del modelo en la app. Para la integración y ejecución del modelo de clasificación de imágenes en el dispositivo, se seleccionó la biblioteca *react-native-fast-tflite*, la cual funciona basada en JSI para la manipulación directa de la API nativa del intérprete C++ de TensorFlow Lite desde el hilo de JavaScript. También ofrece soporte para la aceleración por

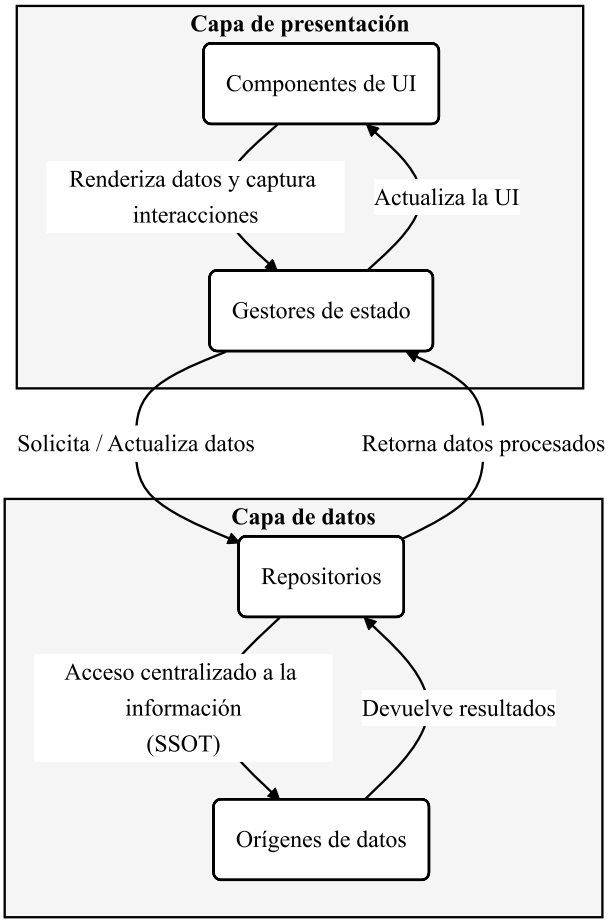
hardware mediante el uso de delegados para asignar los cálculos computacionalmente intensivos del modelo a la GPU del dispositivo o a sus aceleradores de redes neuronales para balancear una mejor velocidad de inferencia y un menor consumo de batería (Rousavy, 2023/2025).

6.3.7 Arquitectura de aplicación móvil. La arquitectura de la aplicación móvil se seleccionó siguiendo los principios de la arquitectura de aplicaciones moderna recomendada por Google para Android, adaptados al paradigma basado en componentes de React Native, que combina capas de presentación y de lógica por su naturaleza funcional basada en componentes, que no se ajusta de manera absoluta a definiciones habituales de arquitecturas como MVC o MVVM. Esta definición de arquitectura también se usa para asegurar la adherencia a las mejores prácticas de la industria para el desarrollo de software móvil (Android Open Source Project, 2025).

La arquitectura se fundamenta en separación de intereses, la conducción de la interfaz de usuario desde modelos de datos, el establecimiento de una única fuente de verdad (SSOT) y un flujo de datos unidireccional (UDF), materializados en una arquitectura en capas que define límites y responsabilidades para cada parte de la aplicación. La estructura de la aplicación se organizó en sus respectivas capas lógicas y se ilustra en la Figura 9.

Figura 9

Arquitectura de la aplicación móvil



Nota. La capa de presentación es la responsable de mostrar los datos de la aplicación en la pantalla y de capturar las interacciones del usuario. Aplicado a React Native, está compuesta por: los elementos de UI, que son los componentes de React para renderizar la información de manera declarativa; y los gestores de estado para exponer las acciones que el usuario puede realizar y manejan la lógica de la interfaz. La capa de datos contiene la lógica de negocio de la aplicación y es responsable de gestionar todas las operaciones de datos, siendo la única fuente de verdad (SSOT) para la información que maneja la aplicación y estructurada mediante el patrón de repositorio que centraliza el acceso a los datos de un dominio específico. A su vez, el repositorio interactúa con una o más fuentes de datos ya sea obteniéndolos de un origen estático o remoto.

CAPÍTULO 3

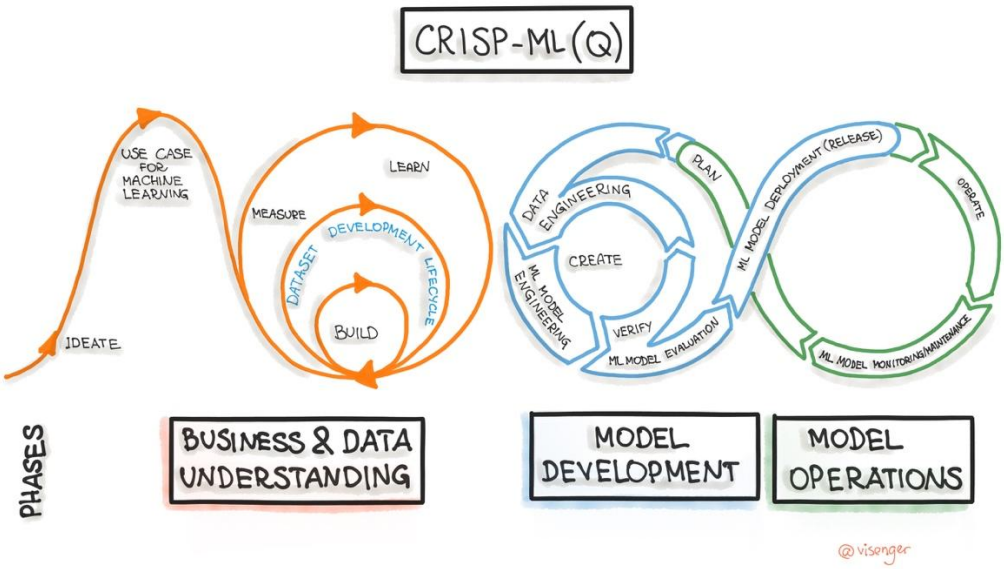
7. METODOLOGÍA

El desarrollo del presente proyecto se enmarca en un enfoque de investigación cuantitativa aplicada, regido por el marco metodológico CRISP-ML(Q) (Cross-Industry Standard Process for Machine Learning with Quality Assurance), el cual es una extensión del marco CRISP-DM y está diseñado específicamente para administrar el ciclo de vida de aplicaciones de Machine Learning, asegurando la reproducibilidad, mitigación de riesgos, y de manera importante, la calidad en cada fase (Studer et al., 2021). Como se ilustra en la Figura 10, el proyecto se articula a través de las fases de desarrollo del modelo: Comprensión del negocio y los datos, preparación de datos, modelado y evaluación, para culminar en una fase de despliegue.

La fase inicial de comprensión del negocio y los datos, que según CRISP-ML(Q) ha sido abordada en los capítulos previos del proyecto al justificar, enmarcar los objetivos y definir la viabilidad del proyecto.

Figura 10

Fases del proceso de desarrollo del modelo CRISP-ML(Q)



Nota. El diagrama ilustra el ciclo de vida del proyecto. Tras la fase inicial de comprensión del negocio y los datos, el proyecto se centra en las fases de desarrollo del modelo y operaciones del modelo. Tomado de *CRISP-ML(Q). The ML Lifecycle Process*, por L. Visengeriyeva, s/f.

La fase de preparación de datos tiene como objetivo transformar los datos brutos recopilados en un conjunto de datos final, limpio y estructurado, listo para la fase de modelado. La calidad del modelo de aprendizaje automático es directamente dependiente de la calidad y cantidad de los datos con los que se entrena. Afirman Studer et al. (2021) que, "si el tamaño de muestreo no es suficiente, el proyecto debe ser terminado o puesto en espera" (p. 6). En esta fase se realizan tareas de obtención de datos, selección y limpieza, construcción de dataset e integración de datos.

En la fase de modelado, se seleccionan, aplican y ajustan las técnicas para alcanzar el objetivo del proyecto, con el propósito de construir un modelo que satisfaga las restricciones y requisitos definidos. Primero se define la técnica de modelado, para este proyecto sería Transfer Learning, luego se diseñan los experimentos de entrenamiento con garantía de fuerte consistencia, reproducibilidad metódica y de resultados, y fácil modificación de hiperparámetros, canalizaciones y variables inherentes.

El objetivo de la fase de evaluación es valorar el grado en que el modelo construido cumple con los objetivos del proyecto y determinar si existen razones para su despliegue (Studer et al., 2021). Por la naturaleza del artefacto la evaluación objetiva valora tal grado, y la subjetiva es pensada para añadir explicabilidad al usuario final o las partes interesadas. El rendimiento del modelo construido se cuantifica utilizando el conjunto de prueba del dataset, y se emplean métricas de clasificación estándar, para este proyecto definidas en la Sección 6.3.4: la matriz de confusión, la exactitud (accuracy), la precisión, la sensibilidad o exhaustividad (recall) y la puntuación F1 (F1-score).

A partir de la evaluación del modelo, se analizan los resultados en el contexto de los criterios de éxito definidos en la fase inicial, comparando las diferentes arquitecturas para seleccionar el modelo óptimo que ofrezca el mejor balance entre precisión y eficiencia. El resultado de esta fase es una decisión informada sobre si el modelo está listo para el despliegue o si es necesario volver a fases anteriores, por ejemplo variando hiperparámetros o preparando más datos.

La fase de despliegue se enfoca en la utilización práctica del modelo en el entorno de producción y campo de aplicación designados, sea un componente de un software más grande con

arquitectura modular, un servicio en un sistema distribuido, un dispositivo de borde, un sistema embebido entre otros. En este proyecto el target es una aplicación móvil, por lo que se define la estrategia para integrar el modelo en la aplicación, selección y disponibilidad del hardware de inferencia (CPU/GPU del dispositivo móvil) y la optimización del modelo para dicha plataforma tanto en hardware, como en software si fuese necesario.

El desarrollo de la aplicación móvil se contempla como etapa en la fase de despliegue dentro de CRISP-ML(Q), entendiendo que es el entorno que ejecuta el modelo de reconocimiento, pero que por su complejidad se desarrollará bajo el marco de trabajo Kanban, con herramienta de gestión de proyectos Jira, y siguiendo prácticas de DevOps en planificación, versionamiento, integración y despliegue continuos para garantizar la entrega de valor de la aplicación móvil como artefacto usable.

Para la fase de monitoreo y mantenimiento, la aplicabilidad se describe y discute en la sección de proyecciones, pensando en un trabajo futuro.

8. DESARROLLO DEL PROYECTO

A continuación se describe todo el desarrollo del proyecto en la secuencia de la metodología definida.

8.1 Preparación de datos

8.1.1 Selección de especies. El primer paso para el dataset es la delimitación. Hasta ahora no se habían definido las especies de flora bogotana con las que se entrenará el modelo, es decir, las que reconocería el modelo. Se debía tener un conjunto manejable y que guarde coherencia con la justificación y contexto del proyecto, un modelo que reconozca absolutamente todas las especies se vuelve inviable. Es mejor tener un modelo funcional para menos especies que uno mediocre para muchas.

Para la búsqueda de especies, se tuvo en cuenta, como primer factor, que las especies fueran comunes, aquellas que son más frecuentes para encontrar dentro de los espacios de la zona urbana y colindante de Bogotá tales como parques, humedales, y las zonas accesibles de los cerros orientales. Como segundo factor, se optaron por escoger especies endémicas y se evitaron incluir especies introducidas, con la salvedad que estas fueran emblemáticas o de interés particular, entendiendo el contexto de la diversidad de ecosistemas encontrados en Bogotá. Después de la búsqueda y revisión, se seleccionó un grupo representativo de especies de interés general, clasificadas por su medio de establecimiento e indicadas en la Tabla 1.

Tabla 1

Especies seleccionadas para el modelo

Nombre científico	Nombre común	Medio de establecimiento
Baccharis latifolia	Chilca	Endémica
Chaetogastra grossa	Sietecueros Rojo	Nativa
Hesperomeles goudotiana	Mortiño	Endémica
Juglans neotropica	Nogal	Nativa
Macleania rupestris	Uva Camarona	Nativa
Meriania nobilis	Amarraboyo	Endémica
Miconia squamulosa	Tuno Esmeraldo	Nativa

Myrcianthes leucoxyla	Arrayán	Nativa
Schoenoplectus californicus	Junco Espadaña	Nativa
Typha domingensis	Enea	Introducida
Weinmannia tomentosa	Encenillo	Nativa

8.1.2 Adquisición de datos. Con las especies seleccionadas, el paso a seguir para la construcción del dataset consistió en obtener las imágenes. Se definió a iNaturalist como fuente principal por: a) la abundancia de imágenes aportadas por los observadores y contribuyentes que conforman el ecosistema de la plataforma; b) la flexibilidad para obtener de manera automática y estructurada los datos de forma masiva, poniendo a disposición diferentes métodos, recursos y documentación, y; c) el proceso riguroso de evaluación de calidad de los datos donde, según la entidad, se aseguran: "precisión, integridad, relevancia e idoneidad de una observación como datos de biodiversidad" (iNaturalist, 2024), lo cual avala aptitud para grado de investigación.

8.1.2.1 Descarga de imágenes de grado de investigación. El primer proceso automatizado en la adquisición de datos fue la descarga de imágenes de grado de investigación, pensadas como contenido principal y mayoritario del dataset por su calidad asegurada, las cuales permitieron aproximar el número de muestras disponible por clase para evaluar el desbalance de clases y mitigarlo en las siguientes etapas. Los filtros y condiciones para tal adquisición fueron los siguientes:

- Las observaciones asociadas a las imágenes deben poseer el grado exclusivo de calidad de investigación "Research grade"
- Las imágenes deben otorgar una licencia Creative Commons de dominio público o permisiva.
- Las observaciones asociadas a las imágenes deben haber sido registradas en Colombia para las especies endémicas y nativas. Para las especies introducidas no se tuvo en cuenta este filtro.

Para dar cumplimiento a estas condiciones, se usó el dataset abierto ofrecido por iNaturalist (iNaturalist contributors & iNaturalist, 2025), ya que es aquel que contiene las observaciones de grado de investigación y licencias permisivas. No se usaron directamente los recursos de

iNaturalist, como su API o su plataforma, siguiendo la recomendación que ellos mismos proporcionan para evitar sobrecargar sus sistemas dada la cantidad de imágenes, por lo cual, se optó por extraer los enlaces de las imágenes de la copia del dataset almacenada en GBIF mediante su API. Las secuencias de consulta a fueron entonces las siguientes:

1. Se obtuvieron los ID de taxón de cada especie a partir del nombre científico en el sistema de clasificación taxonómico *GBIF Backbone*. (GET /species/search)
2. Se obtuvieron las ID, metadatos y fotos de las ocurrencias asociadas a cada ID de taxón, filtradas por el dataset de iNaturalist y con las condiciones mencionadas previamente. (GET /ocurrence/search). Los datos resultantes fueron guardados en un archivo CSV para registrar las selecciones.
3. A partir de los enlaces organizados en el CSV, se creó una función para descargar las fotos de manera paralela para aprovechar al máximo el ancho de banda y los recursos computacionales. La descarga se realizó por carpetas con el nombre de la especie, y se dejó la opción de ajustar las instancias paralelas para ancho de banda más reducido o recursos más limitados.

A nivel de programación, las consultas a la API se hicieron mediante la biblioteca de Python *pygbif*, propia de GBIF para facilitar la operación y procesamiento de las respuestas. La ejecución dio como resultado 10288 imágenes descargadas y distribuidas por especie, como se refleja en la Tabla 2.

Tabla 2

Distribución de imágenes de grado de investigación descargadas por especie

Especie	**Imágenes de grado de investigación**
Baccharis latifolia	1581
Chaetogastra grossa	475
Hesperomeles goudotiana	205
Juglans neotropica	891
Macleania rupestris	1266
Meriania nobilis	385

Miconia squamulosa	745
Myrcianthes leucoxyla	250
Schoenoplectus californicus	2018
Typha domingensis	1945
Weinmannia tomentosa	527
Total	**10288**

8.1.2.2 Descarga de imágenes con identificación necesaria. Después de establecer la primera parte del dataset con imágenes verificadas, el siguiente paso fue descargar de manera automatizada las imágenes asociadas a observaciones que, según la clasificación de calidad iNaturalist "necesitan identificación" (Needs ID), esto es, que necesitan revisión manual de la información asociada a la observación para pasar el filtro de calidad. Se descargaron estas imágenes para prever y subsanar el desbalance de clases que se pudiera encontrar después de hacer la selección y curación de las imágenes que se descargaron en el paso anterior. Porque, si bien, iNaturalist aplica su proceso de calidad, no implica que todas las imágenes tengan una composición apta para el entrenamiento del modelo. El proceso de descarga y las condiciones de adquisición, similares al paso anterior, fueron las siguientes:

- Las observaciones asociadas a las imágenes deben poseer el grado "Necesita identificación"
- Las imágenes deben otorgar una licencia Creative Commons de dominio público o permisiva.
- Las observaciones asociadas a las imágenes deben haber sido registradas en Colombia para las especies endémicas y nativas.
- Solo serán tenidas en cuenta las especies que hayan tenido menos de 1000 imágenes de grado de investigación descargadas en el paso anterior.

Esta vez se usó el API de iNaturalist mediante la biblioteca de Python *pyinaturalist*, que, así como *pygbif*, permitió la operación y procesamiento de respuestas. Las secuencias de consulta a la API fueron las mismas que con la descarga por GBIF. La ejecución dio como resultado 3144 imágenes descargadas y distribuidas por especie, como se detalla en la Tabla 3.

Tabla 3

Distribución de imágenes con identificación necesaria descargadas por especie

Especie	Imágenes con identificación necesaria
Baccharis latifolia	0
Chaetogastra grossa	282
Hesperomeles goudotiana	230
Juglans neotropica	301
Macleania rupestris	0
Meriania nobilis	380
Miconia squamulosa	678
Myrcianthes leucoxyla	1077
Schoenoplectus californicus	0
Typha domingensis	0
Weinmannia tomentosa	196
Total	**3144**

8.1.3 Limpieza y curación de datos. Este proceso se realizó de manera manual y tomó especial parte de tiempo en el proyecto por su importancia en el rendimiento del modelo. Las muestras deben ser lo suficientemente diversas en características visuales, y deben tener una composición adecuada que permita favorecer el entrenamiento, esto es, la especie de interés como sujeto principal, mínima oclusión y cambio en fondos para que el modelo no asocie especies con fondos específicos. Con la diversidad de muestras también se busca favorecer la variabilidad intra-clase, teniendo en cuenta que la representatividad de la especie puede variar según sus diferentes partes (sean hojas, flores/brotes, frutos, tallo/corteza), y la gama de apariencias que la especie pueda tener en los hábitats en los que se encuentre.

En el proceso de adquisición de imágenes es claro el desbalance de clases. La clase mejor representada tiene 38 veces la cantidad de datos de la clase menos representada, lo cual es un problema determinante. Siguiendo con los parámetros de la metodología, Studer et al. (2021) resaltan que un tamaño pequeño de una clase corre el peligro de presentar malos resultados en las

métricas y contribuir al desbalance de clases (p. 6). Indican también que si el tamaño de las muestras no es suficiente, es razón suficiente para quitar la clase (o en otras áreas de aplicación de ML en las cuales solo existe una clase, cancelar el proyecto).

Con las anteriores consideraciones, este proceso entonces tuvo el doble objetivo de: 1) mejorar la calidad visual y la relevancia de cada muestra para el caso de uso de identificación móvil; y 2) mitigar el severo desbalance de clases inherente a los datos de ciencia ciudadana. Se definió un mínimo de 200 y un máximo aproximado de 650 muestras por clase (imágenes por especie).

La limpieza inicial se hizo a partir de las imágenes con grado de investigación. Se aplicaron criterios de selección donde se descartaron imágenes con baja resolución, desenfoque de movimiento, iluminación deficiente, composición confusa donde la planta no era el sujeto principal, oclusión severa, y aquellas cuya identificación taxonómica era dudosa. Adicionalmente, se realizaron recortes en ciertas imágenes para re-encuadrar el sujeto de interés.

Este procedimiento resultó en la reducción del dataset a un total de 4116 imágenes, mejorando significativamente la relación señal-ruido del conjunto de datos y reduciendo en una fase inicial la disparidad entre las clases mayoritarias y minoritarias. Las especies *Hesperomeles goudotiana* y *Myrcianthes leucoxyla* pueden tener un área de mejora introduciendo más muestras. Para este caso, se realizó balanceo a partir de las imágenes de categoría “Necesita identificación”, agregándolas con precaución para no incluir observaciones que no correspondieran a la especie, y con los mismos criterios que se utilizaron en la curación. El dataset finalmente quedó con 4188 imágenes, un peso de ~4.83GB y una línea de base suficiente para empezar a entrenar. La distribución se detalla en la Tabla 4.

Tabla 4

Distribución final de imágenes por especie

Especie	**Imágenes**
Baccharis latifolia	589
Chaetogastra grossa	395
Hesperomeles goudotiana	202

Juglans neotropica	238
Macleania rupestris	621
Meriania nobilis	202
Miconia squamulosa	294
Myrcianthes leucoxyla	205
Schoenoplectus californicus	627
Typha domingensis	443
Weinmannia tomentosa	372
Total	**4188**

Aunque se asumió la posible persistencia de un número marginal de muestras subóptimas, este dataset curado garantizó una base de mejor calidad y validada.

8.1.4 Consolidación del dataset. Luego del proceso de curación y compensación quedaba pendiente la división de los datos en conjunto de entrenamiento, validación y prueba. Siguiendo el estándar y las indicaciones de literatura para datasets de tamaño reducido, se usaron los porcentajes 80-20-20 respectivamente, también había que tener en cuenta que la división debía hacerse de forma estratificada por el desbalance de clases, que aunque reducido, sigue persistente. Para lograrlo se realizó un script que aprovecha la función *train_test_split* de la biblioteca de Python *scikit-learn* que hace la división estratificada de forma automática. La distribución final quedó de 3350 imágenes para el conjunto de entrenamiento, 419 imágenes para el conjunto de validación, y 419 imágenes para el conjunto de prueba.

8.1.5 Versionamiento de los datos. Uno de los pilares de la presente metodología es que se debe asegurar la reproducibilidad de todas las fases del entrenamiento, y sugieren usar prácticas de MLOps. Una de ellas es versionar los datos para llevar un control de qué se cambió, y para este caso fue útil en el proceso de curación de datos, donde se necesitaba tener trazabilidad de qué se borró, que se movió, etc. También se usó para tener un repositorio remoto del dataset pensando en la fase de modelado.

Los datos no pueden ir versionados con el código, por lo que se debe usar un sistema especializado. DVC es el sistema estándar para versionar datos en proyectos de Machine Learning, con flujo y lógica similar a Git (The DVC team and contributors, 2025). Los datos de trazabilidad debían ser versionados con Git, pero las carpetas debían ser excluidas de la trazabilidad de Git mediante el archivo .gitignore.

Al hacer la curación del dataset, cada vez que se terminaba con una especie, se agregaban los cambios y se versionaba el registro de trazabilidad. Al terminar el total, se creó una canalización que comprimía el dataset en su fase final (ya dividido en los conjuntos train-val-test) y lo cargaba al repositorio remoto, definido en un sistema de almacenamiento de objetos en la nube. Con ello, solo bastaba configurar el enlace al repositorio remoto en la máquina donde se quisiera hacer el entrenamiento, descargar el dataset, descomprimir y empezar con el entrenamiento en la siguiente etapa.

8.2 Modelado

La fase de modelado constituyó el núcleo técnico del proyecto, donde se implementó y evaluó sistemáticamente el conjunto de arquitecturas de CNN definidas para desarrollar el clasificador de imágenes, denominado FloraBogNet. Se aplicaron las estrategias de Transfer Learning aprovechando los modelos pre-entrenados en el dataset ImageNet. Para asegurar una comparación equitativa se estableció un protocolo de entrenamiento estandarizado de dos etapas, el cual fue aplicado de manera idéntica a cada una de las cuatro arquitecturas candidatas: MobileNetV2, MobileNetV3-Large, EfficientNetB0 y ResNet50. Cada experimento, comprendiendo ambas etapas para cada arquitectura, se repitió cinco veces, y a su vez cada repetición se ejecutó en un entorno aislado para garantizar la independencia estadística de los resultados y promediar el impacto de la inicialización estocástica de los pesos en las capas clasificadoras.

En la primera etapa se realizó extracción de características para entrenar el clasificador personalizado sobre las representaciones visuales generadas por cada modelo base. Para ello, se instanció cada arquitectura pre-entrenada sin su capa de clasificación superior y se congelaron la totalidad de sus pesos para que no se entrenaran, luego se añadió una "cabeza" clasificadora consistente en una capa de *GlobalAveragePooling2D* para reducir la dimensionalidad de los mapas

de características, una capa de *Dropout* con una tasa de inactivación del 20% para la regularización, y finalmente una capa *Dense* con una función de activación softmax y 11 neuronas de salida, correspondientes a las clases del dataset. El entrenamiento en esta fase se configuró con un optimizador Adam, una tasa de aprendizaje de 0.001 y un tamaño de lote de 32, ejecutándose por un máximo de 30 épocas.

Para mejorar la capacidad de generalización del modelo se aplicó una capa provisional de Data augmentation en tiempo real con volteado horizontal, rotaciones aleatorias de hasta ±72 grados (factor de 0.2) y zoom aleatorio de hasta un 20%, así mismo se implementó mecanismo de parada temprana (early stopping) que monitorizaba la pérdida en el conjunto de validación con una paciencia de 5 épocas, finalizando el proceso automáticamente una vez que el modelo dejaba de mostrar mejoras significativas y restaurando los pesos de la mejor época.

En la segunda etapa se realizó fine-tuning para especializar las capas superiores de cada modelo base a las características visuales específicas del dominio de la flora bogotana. Se inició cargando los pesos del mejor entrenamiento obtenido en la etapa anterior de extracción de características. Posteriormente, se procedió a descongelar las 40 capas superiores de cada modelo base, permitiendo que sus pesos fueran actualizados durante el entrenamiento subsecuente. Para evitar la pérdida catastrófica de conocimiento (catastrophic forgetting), el modelo fue re-compilado manteniendo el optimizador Adam, pero con una tasa de aprendizaje más reducida de 1e-4, y el entrenamiento continuó a partir de la última época de la fase anterior por un máximo adicional de 50 épocas, aplicando nuevamente el mismo mecanismo de parada temprana con idénticos parámetros de paciencia y monitorización.

Para la observabilidad y el registro sistemático de los experimentos, se utilizó TensorBoard como plataforma centralizada de visualización. Para cada ejecución se generó un directorio de logs único donde se registraron de forma automática las métricas por época, tanto para el conjunto de entrenamiento como para el de validación. También se implementó un sistema para registrar la matriz de confusión de validación como una imagen al final de cada época, así como los hiperparámetros de cada ejecución y las métricas finales obtenidas en el conjunto de prueba. Finalmente, los pesos del modelo que alcanzaron el mejor rendimiento en el conjunto de validación durante cada fase fueron guardados automáticamente, creando así un repositorio completo y comparable de todos los artefactos y resultados experimentales. Si en algún momento se quisiera

revisar resultados, analizar el histórico o cargar el estado de cualquier entrenamiento se puede hacer con tales artefactos guardados en cualquier máquina que tenga TensorBoard instalado.

8.2.1 Entorno y flujo de entrenamiento. La ejecución de los experimentos de entrenamiento se llevó a cabo en un entorno de cómputo en la nube, utilizando principalmente instancias de Google Colaboratory equipadas con GPU NVIDIA T4 de 16 GB de VRAM, 2 núcleos de CPU Intel (R) Xeon a 2.0GHz, 12 GB de RAM y almacenamiento variable entre 40 y 60 GB. Para garantizar la reproducibilidad y la portabilidad del proyecto entre este entorno de nube y local se diseñó e implementó un flujo de trabajo sistemático basado en los principios de MLOps, integrando Git para el control de versiones del código fuente y DVC para el versionamiento de datos y artefactos de gran tamaño. Se usó el lenguaje de programación Python, versión 3.11 y un conjunto de bibliotecas y frameworks gestionadas mediante entorno aislado Conda en el desarrollo local, las cuales fueron:

- TensorFlow versión 2.18 (Martín Abadi et al., 2015) y Keras versión 3 (Chollet & others, 2015) para el entrenamiento y operaciones principales de Machine Learning a alto y bajo nivel
- Pandas y NumPy para manipulación y análisis de datos
- Scikit-learn para métricas y utilidades adicionales de Machine Learning
- Matplotlib y Seaborn para visualización de datos

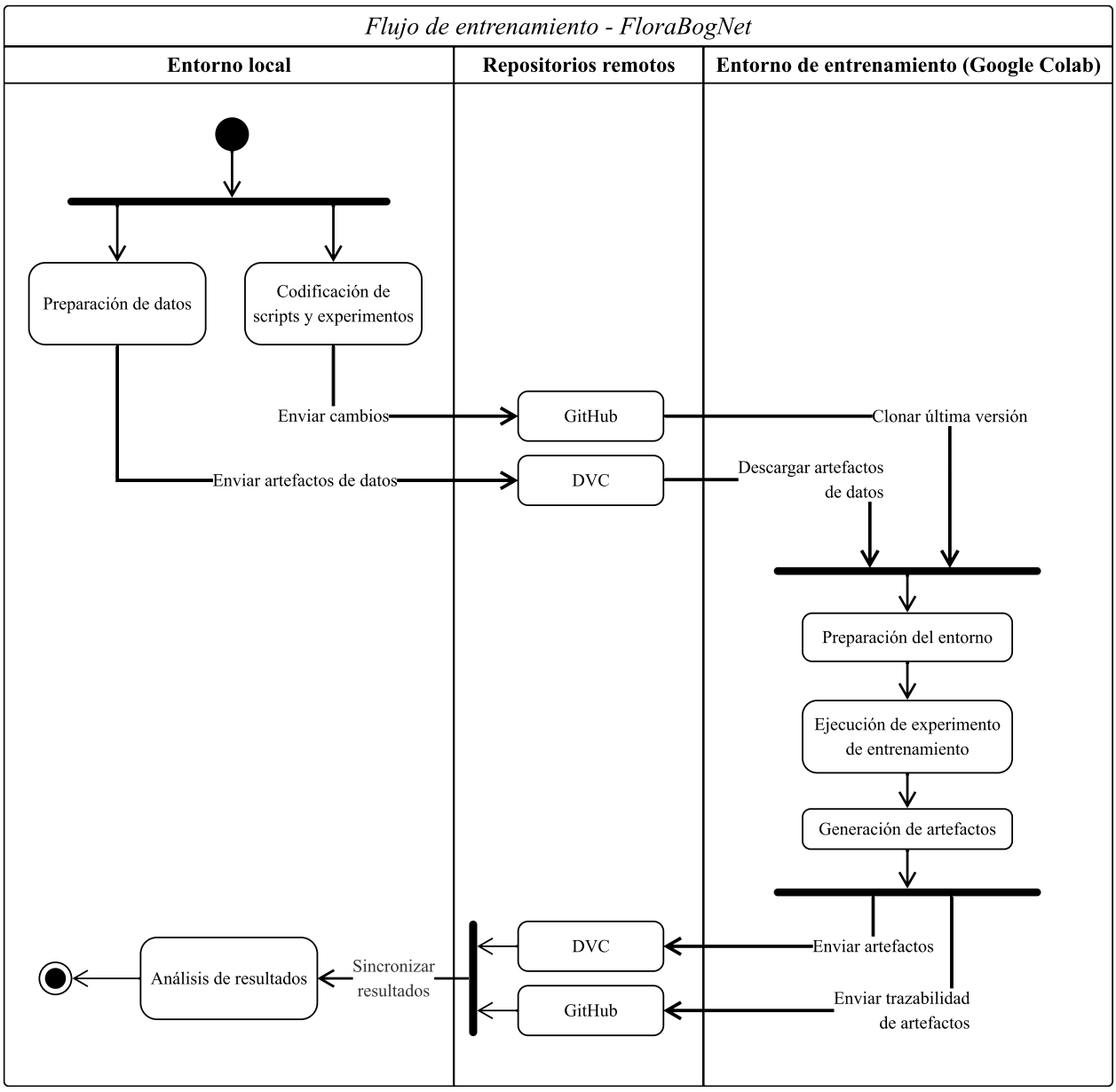
El ciclo de vida de cada experimento se articuló a través de dos repositorios remotos: un repositorio en GitHub para el código fuente y los datos de trazabilidad de DVC (archivos .dvc), y el repositorio remoto de DVC definido en un sistema de almacenamiento de objetos en la nube compatible con S3 para los artefactos de datos pesados. El proceso completo se ilustra en la Figura 11, y fue el siguiente:

1. Tras la finalización de una etapa en entorno local como la preparación de datos o una modificación en el código de las utilidades y/o experimentos de entrenamiento, los cambios se subían a los repositorios remotos correspondientes. El código fuente y los datos de trazabilidad se enviaban a GitHub, mientras que los artefactos de datos versionados –como el dataset por ejemplo– se subían al repositorio DVC remoto.

2. Al iniciar una sesión de trabajo en Google Colaboratory, se ejecutaba un script de preparación que automatizaba primero la clonación de la última versión del código y los datos de trazabilidad desde GitHub, y luego utilizaba DVC para descargar los artefactos de datos necesarios (como el dataset de entrenamiento y los pesos para fine-tuning) desde el servidor remoto en la nube, utilizando credenciales gestionadas de forma segura.
3. Con el entorno sincronizado, se ejecutaba el experimento de entrenamiento correspondiente. Los detalles y partes de cada experimento se detallan en la próxima sección. Este paso fue el que más tiempo y recurso computacional tardaba en cada ejecución.
4. Durante y al finalizar el entrenamiento, se genera un directorio único para el experimento con todos los artefactos resultantes: los pesos del entrenamiento en su mejor época, los logs de entrenamiento con formato de TensorBoard, el reporte de clasificación en formato CSV y la matriz de confusión en formato de imagen.
5. Finalmente, se ejecutaba un script de versionamiento para registrar los artefactos del experimento a DVC, versionar los datos de trazabilidad a Git, y transferir los cambios a sus respectivos repositorios remotos, completando así el ciclo y asegurando la persistencia y disponibilidad de los resultados para el análisis, evaluación y reutilización posteriores.

Figura 11

Diagrama de actividad del flujo de entrenamiento del modelo FloraBogNet



8.2.2 Estructura de los experimentos de entrenamiento. A nivel de estructura de código, el entrenamiento del modelo se adoptó bajo un diseño modular que separa las responsabilidades lógicas del flujo de trabajo de Machine Learning. La orquestación de cada experimento se realizó mediante Jupyter Notebooks, que es el formato en el que se ejecuta el código en Google Colaboratory, y que permite dividir los pasos en bloques ejecutables de manera secuencial con

posibilidad de añadir anotaciones y ver salidas gráficas y de texto por cada bloque. Por otra parte, la lógica de implementación compleja fue abstraída en un conjunto de funciones reutilizables para que al probar una nueva arquitectura o estrategia de entrenamiento se hicieran modificaciones mínimas en los notebooks. Por cada arquitectura y etapa respectivamente se creó un notebook.

El flujo de trabajo dentro de un notebook de entrenamiento para un experimento de primera etapa –la extracción de características– seguía esta secuencia entendida por bloques de código:

1. Se importaban las bibliotecas y se establecían todos los hiperparámetros del experimento, tales como la arquitectura base a utilizar (ej. MobileNetV2), la tasa de aprendizaje, el tamaño del lote, la tasa de dropout, las épocas y la paciencia para el early stopping. En esta misma sección, se generaba un identificador único para la ejecución y se preparaban las rutas para el almacenamiento de artefactos.
2. Se invocaba a una función reutilizable para cargar y preparar los conjuntos de datos de entrenamiento, validación y prueba, devolviendo pipelines de *tf.data.Dataset* optimizados.
3. Se definía la capa de data augmentation para entrenamiento con las alteraciones aleatorias.
4. Otra función se encargaba de construir el modelo de Transfer Learning ensamblando la base pre-entrenada congelada con el clasificador configurado según los hiperparámetros definidos.
5. Una vez construido el modelo se procedía a configurar el entorno de entrenamiento. Una utilidad centralizada se encargaba de generar la lista completa de callbacks de Keras: *ModelCheckpoint*, *EarlyStopping* y los conectores para el logging en TensorBoard. Los callbacks son funciones que se ejecutan al desencadenarse un evento en el entrenamiento.
6. Posteriormente, se compilaba el modelo y se iniciaba el proceso de entrenamiento mediante una clase *Trainer* que encapsulaba la lógica de *.fit()*.
7. Al finalizar el entrenamiento, se ejecutaba la fase final, donde se cargaban los mejores pesos guardados por el checkpoint.
8. Finalmente, se llamaba a un conjunto de funciones de evaluación que calculaban las métricas de rendimiento sobre el conjunto de prueba y se encargaban de generar y guardar todos los artefactos de resultados, como el reporte de clasificación, la matriz de confusión y el registro final de las métricas en TensorBoard.

El flujo para los experimentos de fine-tuning seguía una estructura muy similar, con la adición de pasos intermedios para: cargar los pesos del mejor entrenamiento en la etapa anterior; descongelar las capas superiores de la arquitectura base y; recompilar el modelo con los cambios de hiperparámetros antes de continuar con el entrenamiento.

8.3 Evaluación

8.3.1 Evaluación cuantitativa. Una vez finalizada la fase de modelado, se procedió a la evaluación cuantitativa de los modelos entrenados para materializar los resultados de los procesos anteriores. Este proceso se realizó sobre el conjunto de prueba definido en la fase de preparación de los datos, y que contiene imágenes que ningún modelo había visto hasta ahora. Los resultados consolidados se presentan en la Tabla 5, comparando el rendimiento de los cuatro modelos finalizados tras la etapa de fine-tuning mediante las métricas definidas en la Sección 6.3.4.

Tabla 5

Comparativa de rendimiento de arquitecturas entrenadas en el conjunto de prueba

Modelo	Accuracy	F1 - Macro	F1 - Weighted	Precision	Recall	Número de parámetros
EfficientNetB0	95.47%	94.11%	95.42%	95.90%	94.99%	4,063,662
MobileNetV3-Large	94.03%	93.38%	93.99%	94.92%	93.56%	3,006,923
MobileNetV2	93.08%	91.28%	93.04%	93.95%	92.60%	2,272,075
ResNet50	92.12%	90.48%	92.15%	92.33%	91.89%	23,610,251

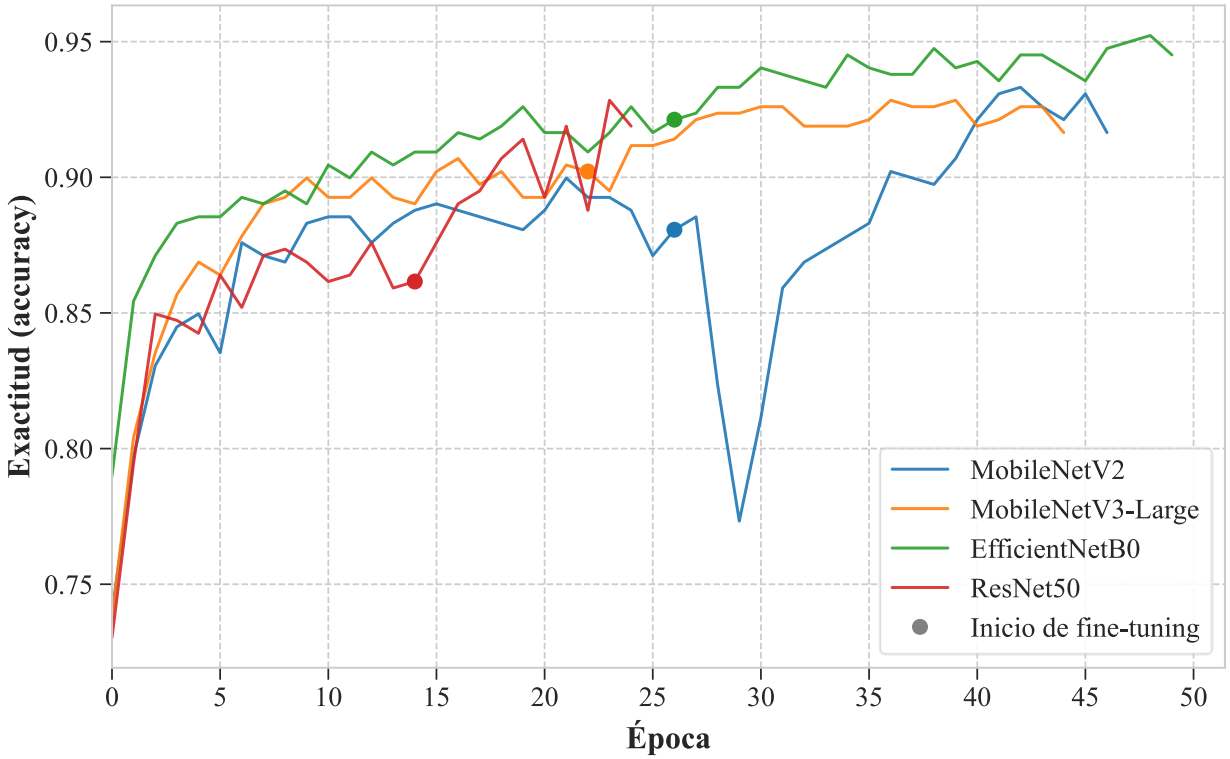
El análisis de las métricas finales evidencia que la arquitectura EfficientNetB0 mostró mayor rendimiento, alcanzando la exactitud más alta con un 95.47% y el F1-Score (Macro) de 94.11%. Le sigue de cerca MobileNetV3-Large, que demostró competitividad con una exactitud del 94.03% y la ventaja de poseer el menor número de parámetros entre los modelos de alto rendimiento. ResNet50, a pesar de ser la arquitectura con mayor capacidad teórica (~23.6 M de parámetros), obtuvo el rendimiento más bajo, sugiriendo una mayor dificultad para generalizar en este dominio específico con el volumen de datos disponible.

Siguiendo con la evaluación retrospectiva en el progreso del aprendizaje, la Figura 12 muestra la evolución de la exactitud en el conjunto de validación a lo largo de las épocas. Todas

las arquitecturas presentan una fase inicial de aprendizaje acelerado. Se ratificó que EfficientNetB0 alcanzó y conservó la mayor precisión en validación durante todo el entrenamiento.

Figura 12

Evolución de la exactitud en el conjunto de validación durante el entrenamiento



Nota. El punto señalado en cada curva indica el inicio del fine-tuning, momento en el que el ajuste de hiperparámetros generó una perturbación temporal seguida de una convergencia hacia un mejor desempeño.

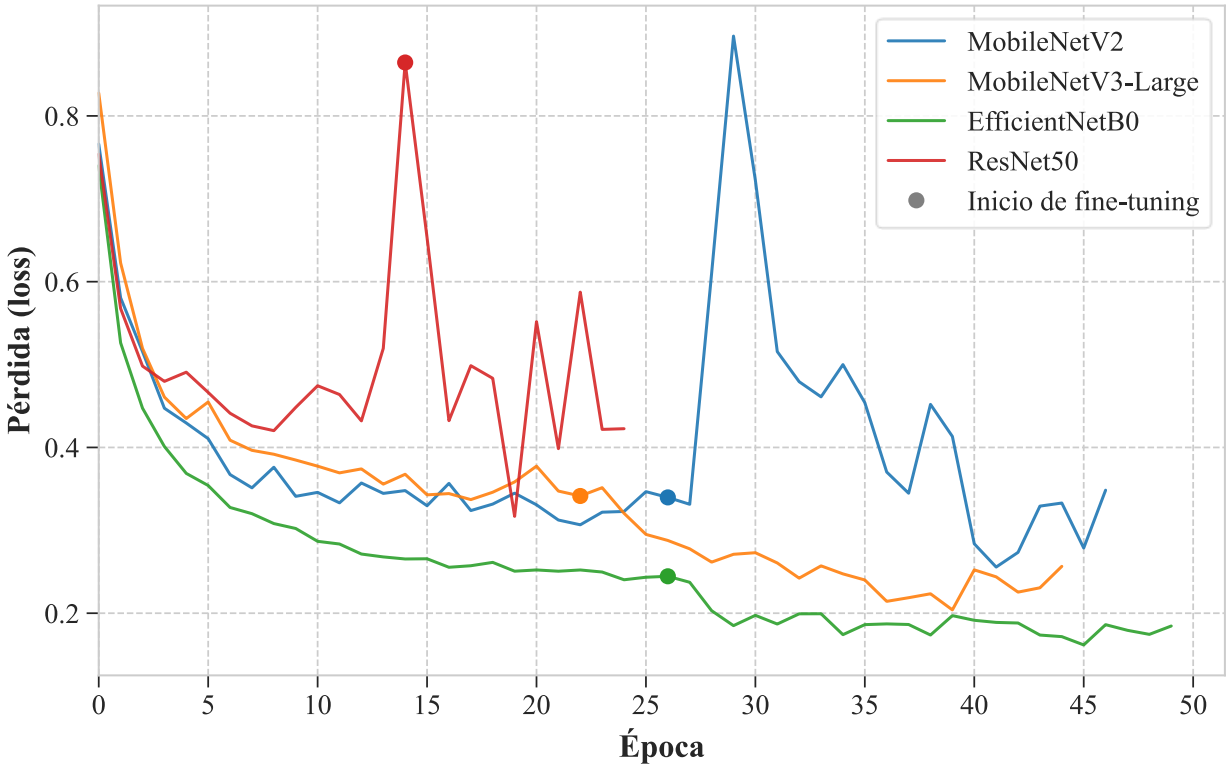
De manera complementaria, la Figura 13 muestra la evolución de la función de pérdida. Se observa una correlación inversa con la precisión, donde EfficientNetB0 converge al valor de pérdida más bajo, demostrando la mayor eficiencia de aprendizaje.

Queriendo detallar el comportamiento de cada modelo, las Figura 14 y Figura 15 presentan las curvas de aprendizaje de entrenamiento y validación de forma individual para la exactitud y la pérdida respectivamente para diagnosticar la capacidad de generalización y detectar indicios de

sobreajuste visualmente. Una vez más, se ratifica que EfficientNetB0 posee mayor rendimiento, estabilidad y con la mejor capacidad de generalización.

Figura 13

Evolución de la función de pérdida en el conjunto de validación durante el entrenamiento



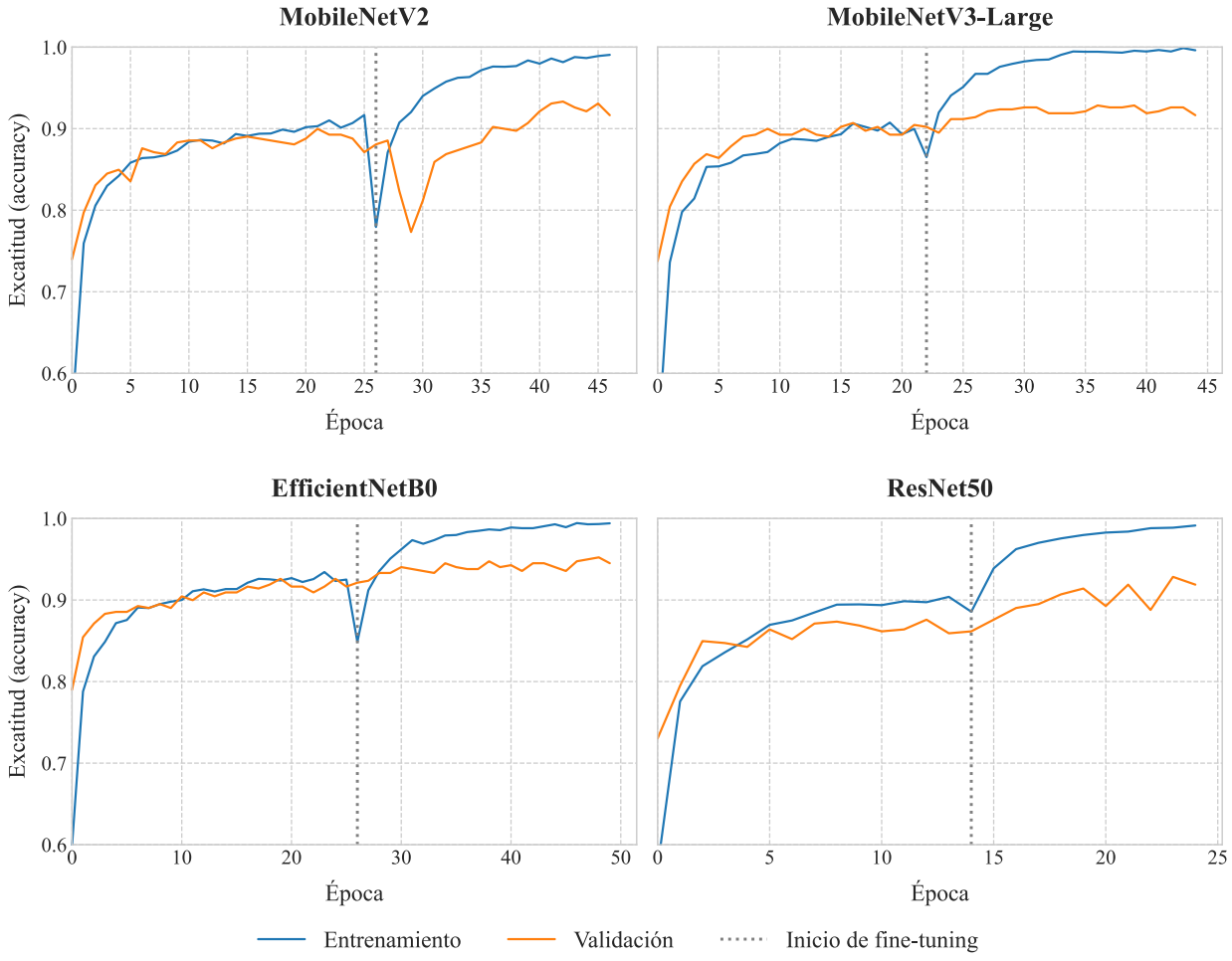
Nota. El pico de pérdida al inicio del ajuste fino es visible en todas las arquitecturas, siendo particularmente pronunciado en ResNet50 y MobileNetV2, indicando inestabilidad al re-entrenar estos modelos o sensibilidad con respecto a los hiperparámetros.

Si bien los análisis de todas estas figuras abordan implícitamente una discriminación entre las arquitecturas entrenadas, el comportamiento del entrenamiento en general entre la mayoría de modelos es estable, beneficiándose en métricas por el cambio de etapa luego de recuperarse de la perturbación mostrada en las gráficas, y validando los mecanismos de regularización implementados. Otro aspecto a destacar es que ResNet50, la arquitectura con menor rendimiento, fue la que aprendió durante menos épocas antes de que se activara el mecanismo de early stopping, y el que mayor tiempo y recurso computacional consumió durante el entrenamiento (50% de la

capacidad de GPU frente al ~20-25% usado por las otras arquitecturas), fuerte indicador de que el tamaño minoritario del dataset se presenta como limitante para arquitecturas de tamaño grande.

Figura 14

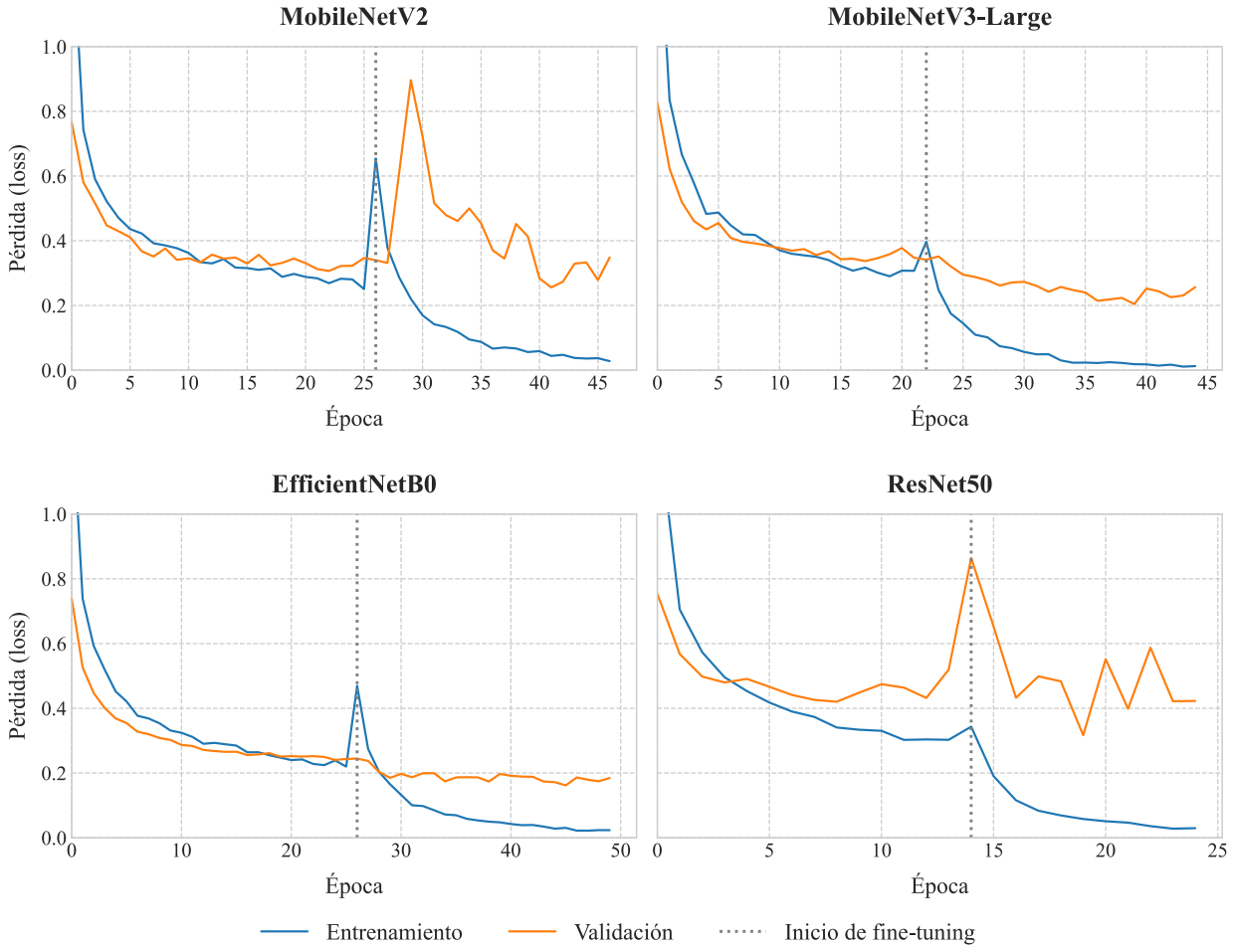
Curvas de exactitud en entrenamiento y validación por arquitectura



Nota. El modelo EfficientNetB0 exhibe comportamiento deseable, ya que la curva de validación sigue de cerca a la curva de entrenamiento a lo largo de todo el proceso, indicando sobresaliente generalización y un sobreajuste mínimo. ResNet50 y MobileNetV2 muestran divergencia más significativa entre ambas curvas hacia el final de cada fase, pudiendo tener dificultades para generalizar. MobileNetV3-Large presenta un comportamiento intermedio pero favorable.

Figura 15

Curvas de exactitud en entrenamiento y validación por arquitectura



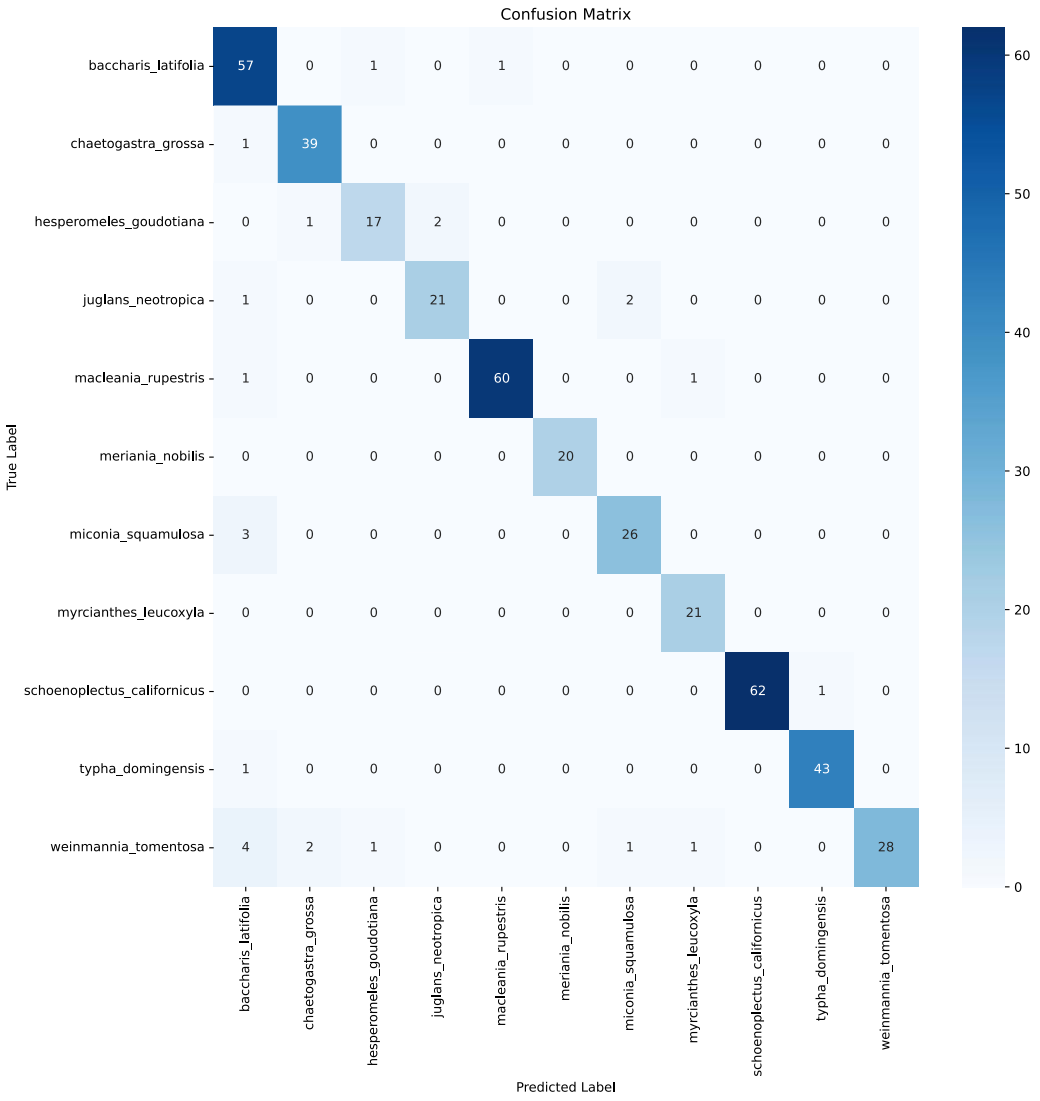
Nota. El modelo EfficientNetB0 muestra pérdida de validación convergiendo al valor más bajo de todos los modelos (~0.2) y manteniéndose estable con mínimas fluctuaciones con alta confianza en sus predicciones sobre datos no vistos. MobileNetV3-Large exhibe un valor de pérdida ligeramente superior pero con una dinámica similar. ResNet50 y, en menor medida MobileNetV2 muestran una mayor volatilidad en la pérdida de validación, especialmente durante la fase de fine-tuning, con que las oscilaciones posteriores indican una menor certeza en sus generalizaciones y propensión a sobreaprendizaje.

Para entender de manera más tangible los aciertos y mejoras de identificación en las especies, la Figura 16 presenta la matriz de confusión del modelo con arquitectura EfficientNetB0

sobre el conjunto de prueba, con menores identificaciones confusas y fuerte concentración de predicciones a lo largo de la diagonal principal.

Figura 16

Matriz de confusión de EfficientNetB0 sobre el conjunto de prueba



Nota. El eje vertical (True Label) representa la clase taxonómica real de las imágenes, mientras que el eje horizontal (Predicted Label) indica la clase asignada por el modelo. Las especies con más confusión y dispersión de errores fueron w*einmannia tomentosa, hesperomeles goudotiana* y *miconia squamulosa*. A pesar de lo anterior, tales confusiones son menores y el modelo concentra un número alto de verdaderos positivos.

8.3.2 Evaluación cualitativa. Para complementar la evaluación cuantitativa y obtener una comprensión cualitativa del comportamiento interno de los modelos, se empleó la técnica de mapas de activación de clases ponderados por gradientes (Grad-CAM) (Selvaraju et al., 2020), la cual genera mapas de calor que resaltan las regiones de una imagen de entrada que fueron más influyentes para que la red tomara una decisión de clasificación específica, respondiendo a la pregunta: ¿en qué partes de la imagen se ha fijado el modelo para llegar a su conclusión?.

Con esta técnica se pretende verificar si el modelo está atendiendo a características botánicas relevantes en lugar de a artefactos del fondo o irrelevantes para la tarea, y facilitar la comparación de las estrategias de inferencia entre diferentes arquitecturas y la depuración visual de fallos de clasificación. La naturaleza de esta técnica es cualitativa, en la medida que muestra el fundamento de clasificación del modelo pero no determina si el criterio visual es bueno o malo, por lo que la observación humana aproxima la fiabilidad del modelo comparando si la atención del modelo está siendo dirigida a elementos de la imagen en las que un humano se fijaría si quisiera identificar una planta.

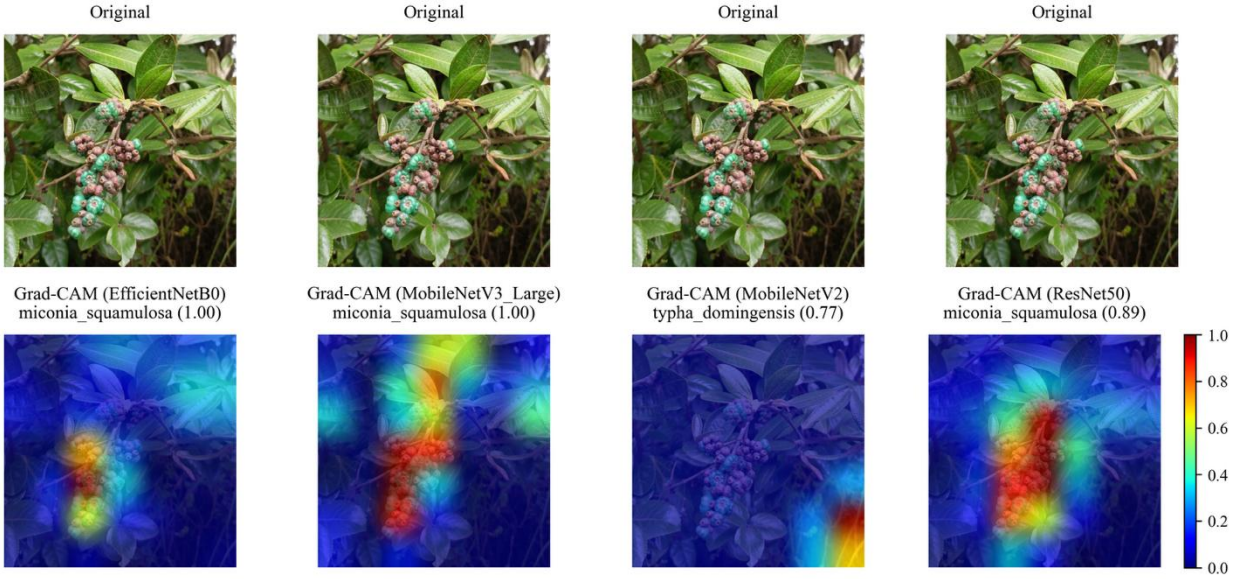
Esta técnica también hace parte de lo que se entiende como un método de IA explicable (XAI), esto es un método que permite explicar por qué se ha producido una predicción concreta en términos comprensibles para el usuario final o las partes interesadas que: o no estén involucradas técnicamente, o no interpreten las métricas cuantitativas (Mersha et al., 2024).

Todas las imágenes que se usaron para esta evaluación fueron descartadas en el proceso de construcción del dataset, por lo que hasta ahora el modelo no había interactuado con ellas, procurando asegurando la transparencia y el comportamiento con imágenes de calidad y composición inferiores a las usadas en el proceso de entrenamiento.

Como primera evaluación se usó la técnica sobre una imagen de prueba de *Miconia squamulosa*, cuyos resultados se presentan en la Figura 17.

Figura 17

Visualización Grad-CAM para las arquitecturas evaluadas sobre una misma imagen



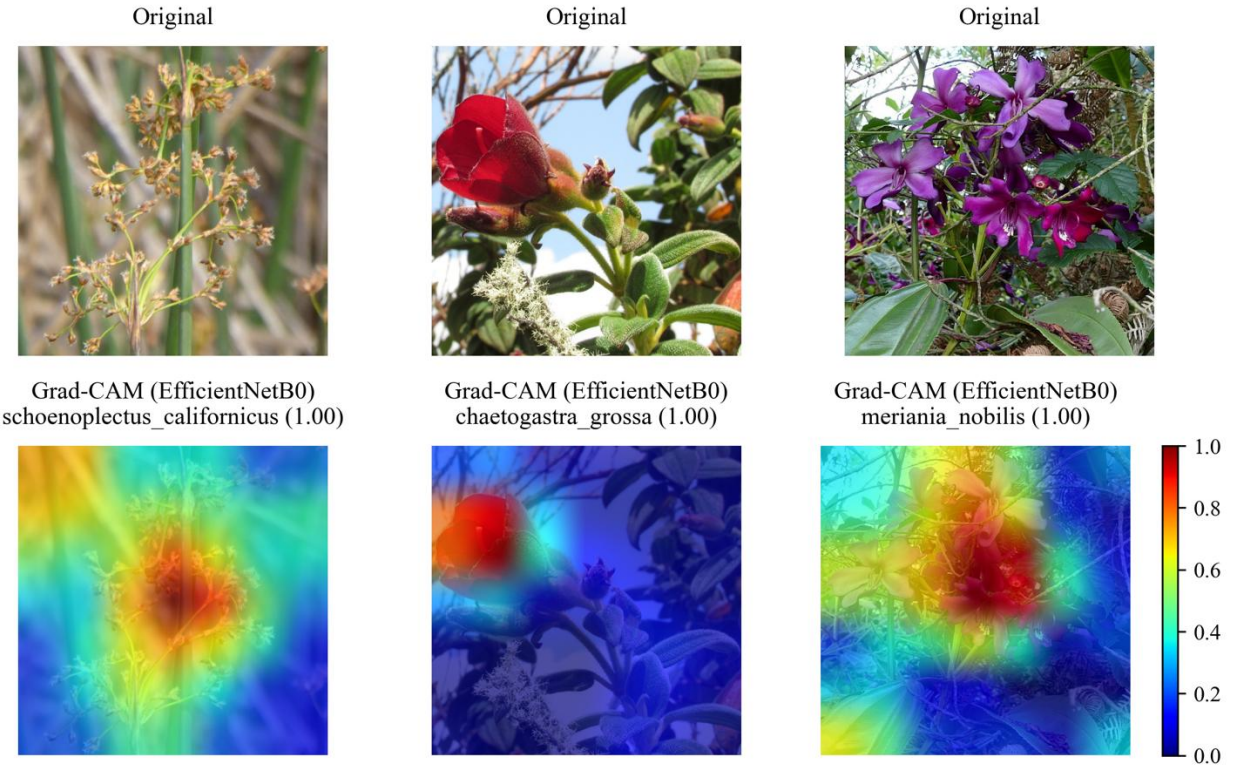
Se observa que tanto el modelo EfficientNetB0 como MobileNetV3-Large clasificaron la imagen correctamente con una confianza del 100%, y sus mapas de activación se concentran las infrutescencias de la planta. El modelo ResNet50, aunque también logró la clasificación correcta, lo hizo con una confianza considerablemente menor (89%) y un mapa de calor más difuso.

El modelo MobileNetV2 falló en la clasificación al identificar erróneamente la imagen como *Typha domingensis*: su mapa de activación es nulo y confunde una planta forestal con una presente en humedales, lo cual alerta de que en la práctica, el modelo entrenado con esta arquitectura no puede ser confiado para el reconocimiento, y por lo consiguiente, debe ser descartado al no responder con los requerimientos.

Después de esta evaluación, se usó la técnica sobre imágenes de diferentes especies con la mejor arquitectura, en este caso EfficientNetB0. Los resultados se presentan en la Figura 18.

Figura 18

Visualización Grad-CAM para EfficientNetB0 evaluada sobre imágenes de varias especies

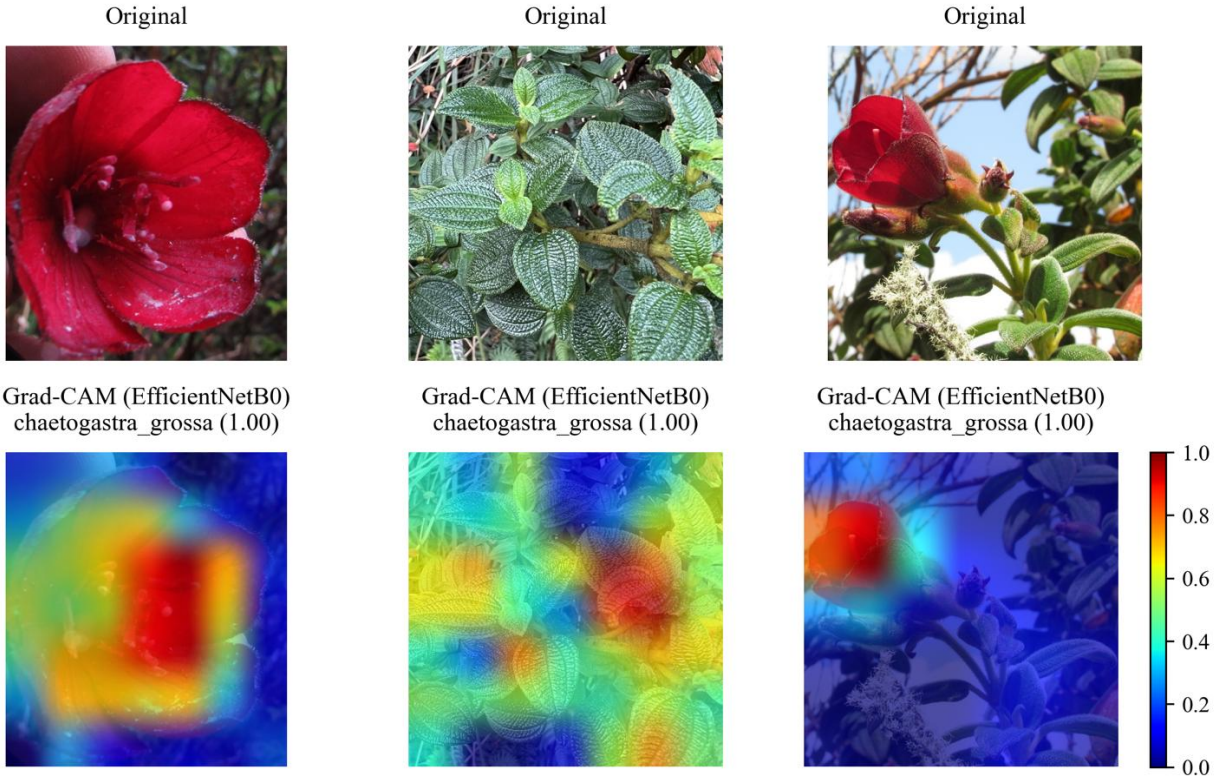


Nota. El modelo clasificó la imagen correctamente con una confianza del 100%, y sus mapas de activación se encuentran de manera concentrada en las inflorescencias de las plantas. El modelo tiene su foco de atención en las características relevantes.

Como paso final y para demostrar la variabilidad intraclase del modelo, la Figura 19 presenta los resultados de la técnica aplicada sobre imágenes de la especie *Chaetogastra grossa* en diferentes partes de la planta.

Figura 19

Visualización Grad-CAM para EfficientNetB0 evaluada sobre diferentes partes de la misma especie



Nota. La técnica se aplicó con una imagen mostrando la flor de sietecueros rojo, otra imagen de sus hojas sin la flor, y por último la planta con su inflorescencia y hojas. El foco de atención prima en la flor, pero si no se encuentra, el modelo basa su predicción en la textura y nervadura distintivas de las hojas. El modelo demostró capacidad para clasificar las especies teniendo en cuenta las diferentes partes y posibles composiciones de la planta.

Con los resultados cuantitativos y cualitativos a partir de la evaluación del modelo se procedió a la siguiente y última etapa contemplada para el proyecto, el despliegue.

8.4 Despliegue: Optimización del modelo

Los modelos de Deep Learning, en su formato de entrenamiento estándar son computacionalmente costosos y su tamaño puede ser prohibitivo para una aplicación móvil teniendo en cuenta las restricciones de espacio de almacenamiento, tiempo de descarga y el consumo de memoria RAM durante la ejecución. Por esta razón, se aplicaron técnicas de optimización ofrecidas por el ecosistema de TensorFlow Lite para reducir el tamaño del modelo y la latencia de inferencia, con una mínima degradación de la precisión (Google, 2024a)

La principal estrategia de optimización empleada fue la cuantización post-entrenamiento, donde se reduce la precisión numérica de los parámetros del modelo que por defecto son números de punto flotante de 32 bits (Float32) a formatos más ligeros. Se optó por la técnica de cuantización de punto flotante de 16 bits que reduce el tamaño del modelo a la mitad al convertir los pesos a formato Float16, con la ventaja de causar una pérdida de precisión mínima o nula. Adicionalmente se integraron metadatos en el archivo del modelo para maximizar compatibilidad de integración con frameworks en los dispositivos (Google, 2024b). Este proceso se realizó utilizando la librería de Python *tflite-suport*.

La evaluación final del rendimiento de los mejores modelos optimizados se realizó sobre el mismo conjunto de prueba para cuantificar el impacto de la optimización y determinar la arquitectura óptima para el despliegue. Los resultados comparativos se indican en la Tabla 6.

Tabla 6

Análisis comparativo del rendimiento post-cuantización de las arquitecturas MobileNetB0 y EfficientNet-B3.

Modelo	**Precisión Float32**	**Precisión Float16**	**Δ Precisión**	**Latencia (ms/img)**	**Tamaño final**
MobileNetV3-Large	94.03%	94.03%	0.00%	~13.7 ms	~6 MB
EfficientNetB0	95.47%	94.51%	-0.96%	~33.7 ms	~8 MB

Nota. La tabla compara el rendimiento de los modelos fine-tuned en su formato original Keras (Float32) contra su versión optimizada a TFLite (Float16). Δ Precisión indica la pérdida de exactitud tras la cuantización. La latencia fue medida en un entorno de CPU estándar.

El modelo MobileNetV3-Large mantuvo su exactitud de prueba en 94.03% sin degradación tras la optimización y alcanzó una velocidad de ~13.7 milisegundos por imagen, siendo más de 2.4 veces más rápido que EfficientNetB0. Por el balance del compromiso entre alta precisión, tamaño de archivo reducido (~6 MB) y una velocidad de inferencia superior, se eligió como arquitectura adecuada para la aplicación móvil.

8.5 Despliegue: Aplicación móvil

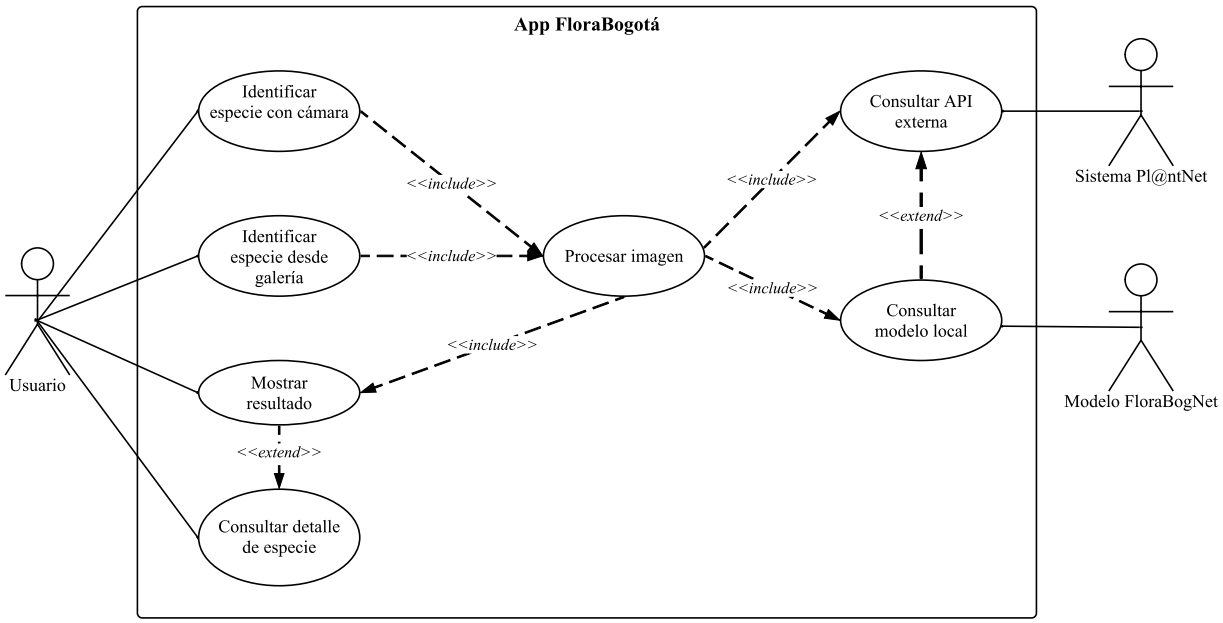
En esta sección se detallará el proceso de diseño y desarrollo de la aplicación, así como la integración del modelo en las funcionalidades propuestas.

8.5.1 Definición de funcionalidades. La aplicación fue diseñada para operar en torno a tres funcionalidades principales centradas en el objetivo de que se creara un objeto de aprendizaje móvil situado donde se proporciona una función técnica y a su vez se enriquezca el conocimiento del usuario en el contexto de su entorno inmediato (Naveed et al., 2023). Estas funcionalidades son: la identificación automatizada de especies, la exploración de catálogo de flora, y la gestión de observaciones personales. Cada una de estas funcionalidades principales se describe en detalle a continuación junto con su caso de uso.

8.5.1.1 Identificación automatizada de especies. La funcionalidad central de la aplicación es su capacidad para identificar especies de flora a partir de imágenes, y se diseñó pensando en maximizar tanto la velocidad y disponibilidad como el alcance de la identificación. El primer nivel es la inferencia local, que utiliza el modelo entrenado para reconocer las 11 especies definidas de forma instantánea y sin necesidad de conexión a internet. Para extender la utilidad de la aplicación más allá de este catálogo inicial, se implementó un segundo nivel de identificación. Entonces, cuando el modelo local no logra una predicción con un umbral de confianza definido, o para asistir al usuario en la identificación de otras especies que hagan parte de la flora bogotana, la aplicación consulta automáticamente al servicio externo de Pl@ntNet que analiza la imagen y devuelve una lista de posibles coincidencias presentadas al usuario como sugerencias. La Figura 20 ilustra el caso de uso para esta funcionalidad.

Figura 20

Diagrama de caso de uso CU-01: Identificación automatizada de especies

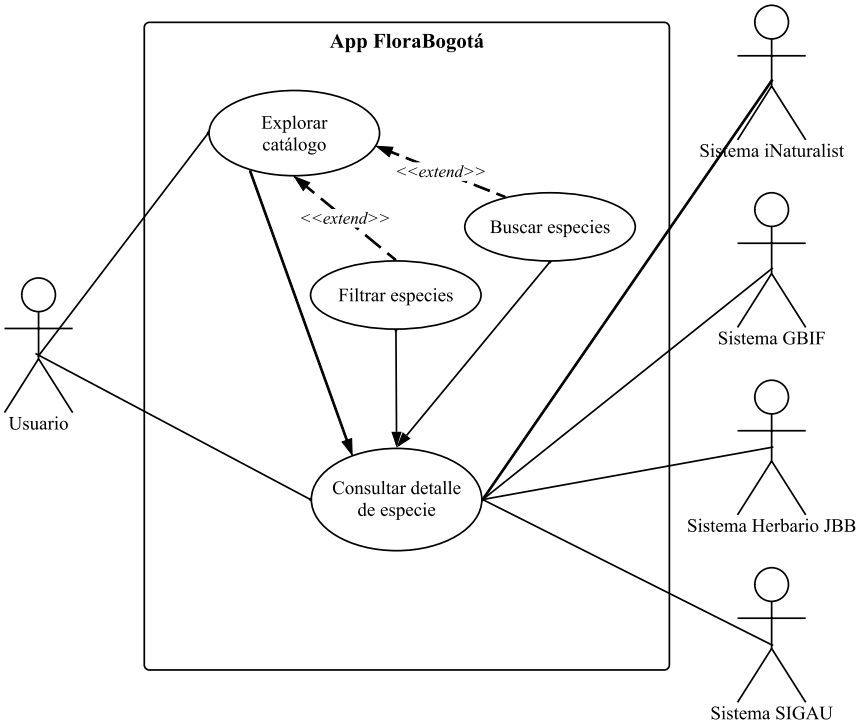


8.5.1.2 Exploración de catálogo de flora. Esta funcionalidad sirve como una enciclopedia de referencia sobre la flora bogotana, que contiene información sobre aproximadamente 350 especies relevantes para el ecosistema de la ciudad, incluyendo tanto especies nativas como exóticas a partir de los hallazgos de información de las diferentes entidades distritales (JBB, 2019, 2023, 2024; Quimbayo-Ruiz, 2016), y permitiendo al usuario aprender y explorar la flora bogotana de manera proactiva sin necesidad de tener una planta física para identificar, y presentándole descripción de la planta, imágenes y otros datos de interés. La

Figura 21 ilustra el caso de uso para esta funcionalidad.

Figura 21

Diagrama de caso de uso CU-02: Exploración de catálogo de flora

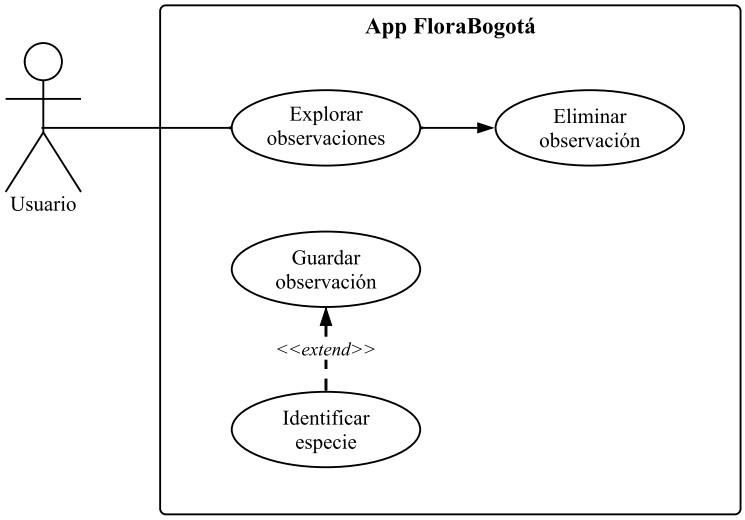


8.5.1.3 Gestión de observaciones personales. Para fomentar el compromiso a largo plazo y la creación de un vínculo personal con el proceso de aprendizaje, se implementó una funcionalidad de observaciones personales que permite al usuario guardar un registro de sus identificaciones exitosas. Cada registro almacena la imagen capturada por el usuario, el nombre de la especie identificada, la fecha de la observación y opcionalmente los datos de geolocalización. La

Figura 22 ilustra el caso de uso para esta funcionalidad.

Figura 22

Diagrama de caso de uso CU-03: Gestión de observaciones personales



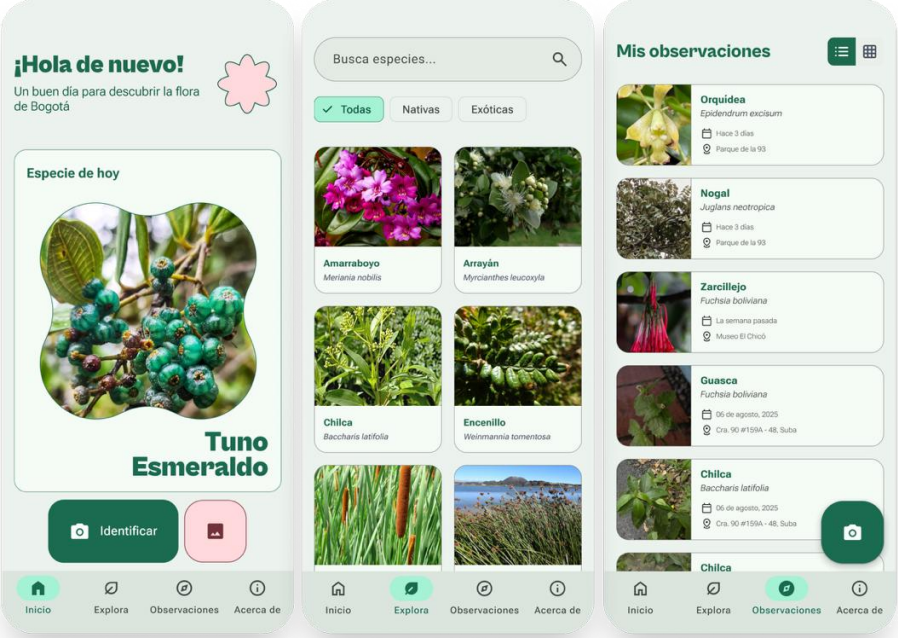
8.5.2 Diseño de interfaz y experiencia de usuario (UI/UX). El diseño de la interfaz y la experiencia de usuario UI/UX se fundamentó en los principios del sistema de diseño Material 3 Expressive, secuela del sistema de diseño propuesto por Google para interfaces móviles y de uso general.

Como documentan Bentley et al. (2025) para la investigación de Google Design, el principio del sistema de diseño “expresivo” busca deliberadamente crear interfaces que conecten con las personas a un nivel emocional, utilizando estratégicamente el color, la forma, el tamaño y la tipografía para comunicar la función y guiar la atención del usuario. Tales características son pertinentes para una aplicación educativa, ya que una interfaz que inspira una respuesta emocional positiva y reduce la carga cognitiva puede fomentar de manera significativa la curiosidad, el compromiso y la retención del conocimiento. En la misma investigación se demuestra que los diseños expresivos no solo son preferidos por los usuarios, sino que también mejoran la usabilidad, permitiendo localizar elementos clave de la interfaz hasta cuatro veces más rápido. Se adoptó entonces esta filosofía para asegurar que la aplicación, además de funcional fuera una herramienta que los usuarios encontraran agradable y motivadora para usar repetidamente. El wireframe de alta fidelidad realizado con los principios mencionados se referencian en la Figura 23 y

Figura 24.

Figura 23

Wireframe de alta fidelidad para app móvil - pantallas principales



Nota. La pantalla de inicio sirve como un portal de bienvenida con una tarjeta central de "Especie del Día" para captar inmediatamente la atención y fomentar el descubrimiento. Los botones de acción principal permiten realizar la identificación desde la cámara o imagen de la galería. La pantalla de exploración implementa el catálogo de flora en vistas de cuadrícula con funcionalidades de búsqueda y filtrado. La pantalla de observaciones organiza el diario personal del usuario en vista de lista y tarjetas con botón de acción que sirve como un recordatorio contextual para añadir nuevas identificaciones.

Figura 24

Wireframe de alta fidelidad para app móvil - detalle de especie

Nota. La pantalla de detalles de especie es mostrada al realizar una identificación exitosa o buscar una especie, y presenta la información de manera editorial con contenido segmentado en tarjetas.

8.5.2 Desarrollo de la aplicación. La aplicación se desarrolló bajo el marco de trabajo ágil Kanban visualizando el trabajo en un tablero digital, limitando las tareas en progreso para concentrar el esfuerzo y adaptar las prioridades. Esta gestión del proyecto se complementó con una filosofía de DevOps, aplicando prácticas de automatización y mejora continua al ciclo de vida del desarrollo, desde la planificación hasta la integración del código.

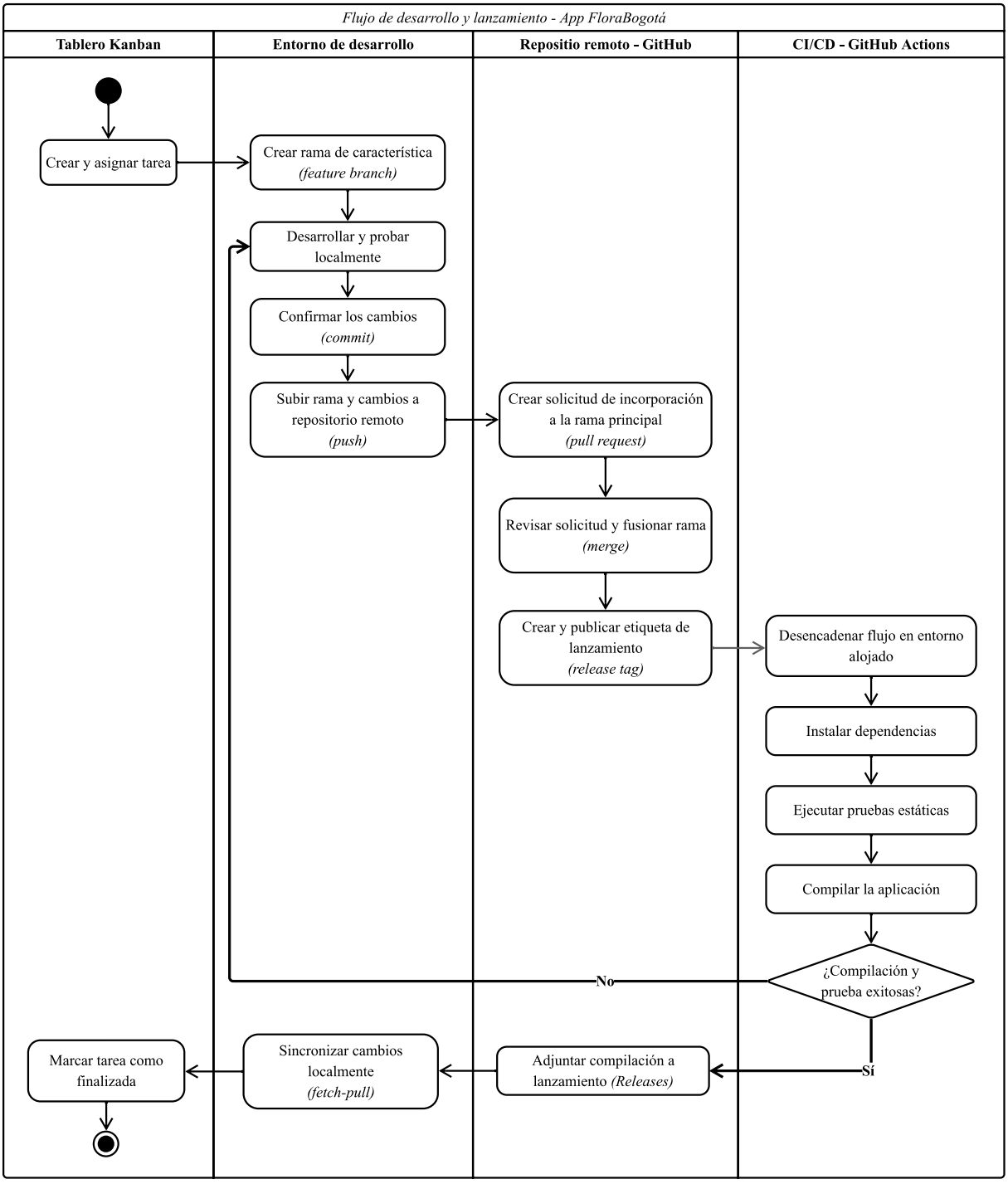
Como se detalla en la Sección 6.3.6, el stack tecnológico principal consistió en React Native como framework para la construcción de la aplicación nativa multiplataforma, y el framework Expo como capa de extensibilidad para incluir características aplicables al dispositivo –por ejemplo, el acceso a la funcionalidad de la cámara o la galería– y como abstracción para simplificar el proceso de desarrollo y compilación multiplataforma. Se complementó con bibliotecas para animación, creación de componentes, navegación, entre otros.

La decisión de utilizar React Native con Expo se justificó por la capacidad de mantener un único código fuente en TypeScript para ambas plataformas móviles y reducir tiempo y esfuerzo de desarrollo y mantenimiento. Los proyectos móviles en etapa temprana suelen priorizar Android sobre iOS por la facilidad de acceso a herramientas de desarrollo, pero limitan bastante el alcance al que la aplicación pueda llegar en un futuro. El rendimiento frente a bases de código nativas (Kotlin para Android, Swift para iOS) no se degrada significativamente teniendo en cuenta la arquitectura que usa el framework (Explicado en la Sección 6.3.6.1)

El flujo de desarrollo se alineó con las prácticas de DevOps mediante una estrategia de ramificación basada en características aplicada al sistema de control de versiones Git: 1) Cada tarea o funcionalidad identificada en el tablero Kanban, se tradujo en una rama de característica dedicada, creada a partir de la rama principal estable. Todo el desarrollo de la funcionalidad se realizaba de forma aislada en esta rama. 2) Una vez completada y probada, la característica se integraba de nuevo en la rama principal a través de una solicitud de incorporación (pull request). Si bien la aplicación se gestionó por un único desarrollador, se facilitó la revisión del código. Se implementaron flujos de trabajo de integración continua (CI) y entrega continua (CD) mediante GitHub Actions. Cada vez que se creaba una nueva versión de lanzamiento etiquetada (reléase tag), se disparaba automáticamente una canalización que ejecutaba una secuencia de trabajos: 1) la instalación de dependencias; 2) la ejecución de pruebas estáticas de código para detectar errores y asegurar la consistencia del estilo, y finalmente; 3) la compilación de la aplicación. Si todos los pasos eran exitosos, la canalización procedía a 4) crear un artefacto de lanzamiento versionado y disponible directamente en el repositorio de GitHub para su descarga e instalación en los dispositivos móviles. Todo este proceso se ilustra de mejor manera en la Figura 25.

Figura 25

Diagrama de actividad - flujo de desarrollo y lanzamiento de la aplicación móvil



8.5.3 Pruebas de la aplicación. La fase de pruebas se diseñó para validar la correctitud, usabilidad y tolerancia a fallos de la aplicación, asegurando que el artefacto de software cumpliera con las funcionalidades definidas mediante pruebas de caja blanca para verificar la calidad estructural del código y pruebas de caja negra para validar el comportamiento funcional desde la perspectiva del usuario final.

Las pruebas de caja blanca se enfocaron en la calidad interna y la integridad del código fuente. Dada la naturaleza del proyecto, estas pruebas se centraron principalmente en métodos estáticos y automatizados integrados en el flujo de trabajo de desarrollo. Se configuró un conjunto de herramientas de análisis de código estático con ESLint y Prettier, dentro del entorno de desarrollo, y ejecutadas automáticamente como un paso obligatorio en la canalización de lanzamiento. También se incluyó una batería de pruebas unitarias que instancian el modelo para su inferencia, y validan el funcionamiento de los componentes y funciones más utilizados en la base de código.

Las pruebas de caja negra se diseñaron para verificar el comportamiento funcional de la aplicación desde el punto de vista del usuario, sin conocimiento de la implementación interna. Se ejecutó un conjunto de casos de prueba manuales sobre un parque de dispositivos físicos seleccionados para representar una diversidad de sistemas operativos, tamaños de pantalla y capacidades de hardware. La validación se realizó en los dispositivos físicos detallados en la Tabla 7, cubriendo tanto la plataforma Android como iOS, así como factores de forma de teléfono y tableta, y con diversos propósitos aprovechando la variedad de hardware y versión de sistema operativo.

Tabla 7

Dispositivos físicos utilizados en la fase de pruebas

Dispositivo	Sistema operativo	Uso para las pruebas
Google Pixel 6 Pro	Android 16	Validación del delegado de inferencia en TPU, compatibilidad con la versión más reciente de Android, dispositivo principal de pruebas de rendimiento.

Xiaomi Mi A3	Android 13	Verificación de la adaptabilidad de la UI a diferentes densidades de píxeles y factores de escala de pantalla.
Galaxy J7 (2016)	Android 7.1	Pruebas de compatibilidad regresiva en dispositivos de gama baja y con la versión más antigua de Android soportada por Expo.
iPad 10.9" 10ª gen	iPadOS 26	Validación de la interfaz de usuario y la experiencia en un factor de forma de tableta.
iPhone 13	iOS 26	Dispositivo principal de pruebas de rendimiento y funcionalidad en el ecosistema iOS moderno. Validación de delegado de inferencia en CoreML con Neural Engine.
iPhone XS Max	iOS 18.6	Pruebas de compatibilidad y rendimiento en un dispositivo iOS más antiguo con pantalla de gran tamaño.

Se definió un conjunto de casos de prueba para cubrir los flujos de usuario críticos de la aplicación. Los resultados se registraron y analizaron para identificar y corregir defectos. A continuación, se resumen los casos de prueba más representativos y los hallazgos:

- Se probó la identificación de las 11 especies del modelo local con imágenes de alta calidad. El sistema clasificó correctamente todas las especies con una confianza superior al 90% en todos los dispositivos.
- Se utilizaron imágenes de especies no incluidas en el modelo local. Se verificó que el sistema, tras obtener una baja confianza del modelo local, consultara automáticamente la API de Pl@ntNet y mostrara correctamente la lista de sugerencias.
- Se probó la búsqueda y el filtrado en el catálogo de 348 especies, así como la correcta carga de datos en la pantalla de detalles.

- Se validó el flujo de guardar, visualizar y eliminar observaciones en el diario de campo, confirmando la correcta persistencia de los datos en el almacenamiento local del dispositivo.
- Se evaluó la consistencia visual y la correcta disposición de los componentes en todos los dispositivos de la Tabla 7

Durante la ejecución de las pruebas, la mayoría de los casos de uso se completaron exitosamente. Sin embargo, se identificaron y subsanaron dos defectos menores relacionados con la adaptabilidad de la interfaz:

- En el dispositivo de gama baja se detectó un desbordamiento visual en los nombres científicos largos dentro de las tarjetas del catálogo. El defecto fue corregido implementando una lógica de truncamiento de texto y ajuste dinámico del tamaño de la fuente.
- En el iPad, la vista de cuadrícula de las pantallas "Explora" y "Observaciones" mostraba solo dos columnas, resultando en un uso ineficiente del espacio. Se implementó una lógica de diseño responsivo para aumentar el número de columnas a tres o cuatro en pantallas con un ancho superior a un umbral predefinido.

CAPÍTULO 4

10. CONCLUSIONES

A lo largo del desarrollo del sistema de reconocimiento, se derivan las siguientes conclusiones:

Se concluye que el marco metodológico CRISP-ML(Q) es un estándar efectivo para guiar el ciclo de vida de un proyecto de aprendizaje automático por su componente de aseguramiento de calidad en cada fase y la definición de artefactos y criterios de éxito claros con fundamentación en evidencia cuantitativa.

Se determinó que es viable construir un dataset especializado y de alta calidad para la flora de Bogotá a partir de fuentes de ciencia ciudadana, validando que tal cantidad de datos, al ser complementada con técnicas de data augmentation es suficiente para prevenir el sobreaprendizaje y entrenar redes neuronales profundas en un dominio con datos limitados.

Se concluye que, para un dataset de tamaño acotado, las arquitecturas optimizadas para la eficiencia computacional superaron en rendimiento a las de mayor capacidad teórica. El modelo EfficientNetB0 alcanzó la precisión más alta (95.47%), superando a MobileNetV3-Large (94.03%), MobileNetV2 (93.08%) y ResNet-50 (92.12%), demostrando que el menor número de parámetros de las arquitecturas ligeras favorece una mejor generalización y confirmando que la selección de la arquitectura debe estar en función de la escala de los datos.

Se validó la viabilidad del despliegue de un modelo de alto rendimiento en un dispositivo móvil y se determinó que el framework Expo sobre React Native es una solución tecnológica eficiente para el prototipado y desarrollo de aplicaciones multiplataforma al abstraer la configuración nativa de cada sistema operativo para validar la integración del modelo y las funcionalidades de la aplicación en un tiempo de desarrollo ajustado.

11. RECOMENDACIONES

Para trabajos futuros de similar naturaleza se presentan las siguientes consideraciones:

La preparación del dataset es fundamental en el desarrollo de un modelo como el de este trabajo y puede determinar su éxito o inviabilidad. De igual manera, es el área de mejora más importante para trabajos futuros, pensando en agregar más especies que el modelo pueda reconocer. El reconocimiento de especies de flora siempre va a tener un limitante de cantidad de datos por especie que se puedan conseguir, por lo que al pensar en cambiar de arquitectura o estrategia de entrenamiento hay que tenerlo presente.

En cuanto a los resultados, este trabajo también se puede extender: evaluando la precisión de reconocimiento respecto a otras aplicaciones comerciales y de uso general; y la usabilidad de la aplicación en entornos educativos segmentados. Otra área de posible expansión fuerte del proyecto es la articulación con las entidades del Distrito, sea el Jardín Botánico o la Secretaría de Ambiente, ya que se justifica y enmarca totalmente con las líneas de acción de sus políticas públicas.

Un componente importante que puede ayudar a agilizar e incrementar los resultados del proyecto es contar con un equipo multidisciplinar que se pueda dedicar a cada fase: Unos a los datos, otros al entrenamiento, otros a la aplicación móvil. En el presente proyecto, al haber una sola persona, los resultados se tardaron más en materializar. A pesar de lo anterior, el proceso se adapta perfectamente a un incremento de personas sin afectar el ciclo planteado, porque la metodología busca asegurar calidad en cada etapa.

En el apartado de software, la compatibilidad entre bibliotecas y frameworks para machine Learning suele ser conservadora, y las versiones que se quieran usar se deben escoger con base a la compatibilidad entre ellas, con previsión de que puedan fallar y haya que repetir los procesos. Para este proyecto la compatibilidad más conflictiva fue entre Keras y TensorFlow Lite, ya que esta última ofrece utilidades que no están en mantenimiento activo y solo soportan versiones antiguas.

REFERENCIAS

Acuerdo No. 009 de 2021: Por el cual se adoptan las líneas translocales de la Universidad de Cundinamarca, 009 Acuerdo (2021). https://www.ucundinamarca.edu.co/investigacion/index.php/lineas-de-investigacion

Android Open Source Project. (2025, febrero 10). *Guide to app architecture | App architecture*. Android Developers. https://developer.android.com/topic/architecture

Bentley, F., Feldman, J., Schmidt, L., Pielot, M., & Gilbert, M. (2025). *Expressive Design: Google's UX Research*. https://design.google/library/expressive-material-design-google-research

Berzal, F. (2018). *Redes Neuronales & Deep Learning*.

Campbell, N., Peacock, J., & Bacon, K. L. (2023). A repeatable scoring system for assessing Smartphone applications ability to identify herbaceous plants. *PLOS ONE*, *18*(4), e0283386. https://doi.org/10.1371/journal.pone.0283386

Chollet, F. & others. (2015). *Keras*. https://keras.io

Decreto 555 de 2021: Por el cual se adopta la revisión general del Plan de Ordenamiento Territorial de Bogotá D.C., Pub. L. No. 555, Registro Distrital (2021). https://www.alcaldiabogota.gov.co/sisjur/normas/Norma1.jsp?i=119582

Expo. (2025). *Expo Documentation*. Expo Documentation. https://docs.expo.dev

Flores, G. M., Guzmán, I. O. J., Escalante, G. G., & López, A. B. (2024). Reconocimiento de flores endémicas de la región de Misantla a través de una red neuronal convolucional usando aprendizaje por transferencia. *Ciencia Latina Revista Científica Multidisciplinar*, *8*(6), Article 6. https://doi.org/10.37811/cl_rcm.v8i6.15136

GBIF. (s/f). *What is GBIF?* Recuperado el 18 de agosto de 2025, de https://www.gbif.org/what-is-gbif

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT press.

Google. (2024a, agosto 30). *Model optimization | Google AI Edge*. Google AI for Developers. https://ai.google.dev/edge/litert/models/model_optimization

Google. (2024b, agosto 30). *Post-training quantization | Google AI Edge*. Google AI for Developers. https://ai.google.dev/edge/litert/models/post_training_quantization

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition* (No. arXiv:1512.03385). arXiv. https://doi.org/10.48550/arXiv.1512.03385

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). *Searching for MobileNetV3* (No. arXiv:1905.02244). arXiv. https://doi.org/10.48550/arXiv.1905.02244

iNaturalist. (2024, julio 30). *What is the Data Quality Assessment and how do observations qualify to become "Research Grade"?* iNaturalist Help. https://help.inaturalist.org/en/support/solutions/articles/151000169936-what-is-the-data-quality-assessment-and-how-do-observations-qualify-to-become-research-grade-

iNaturalist contributors & iNaturalist. (2025). *iNaturalist Research-grade Observations* [Ocurrence dataset]. iNaturalist.org. https://doi.org/10.15468/ab3s5x

Jardín Botánico de Bogotá José Celestino Mutis. (2019, octubre 19). *Herbario JBB en línea*. https://herbario.jbb.gov.co/

Jardín Botánico de Bogotá José Celestino Mutis. (2023, noviembre 15). *CONOZCA EL SISTEMA DE GESTIÓN PARA EL ARBOLADO URBANO*. ArcGIS StoryMaps. https://storymaps.arcgis.com/stories/4261e58f91fe416d935c8f15db4c1cec

Jardín Botánico de Bogotá José Celestino Mutis. (2024, noviembre 30). *Flora de Bogotá—Subdirección científica—Jardín Botánico José Celestino Mutis*. https://florabog.jbb.gov.co/

Jones, H. G., & Jones, A. J. (2025). Application and pitfalls of the use of plant ID apps for urban flora and citizen science studies. *Plant Ecology & Diversity*, *0*(0), 1–9. https://doi.org/10.1080/17550874.2025.2476938

Kaur, P., Harnal, S., Gautam, V., Singh, M. P., & Singh, S. P. (2023). A novel transfer deep learning method for detection and classification of plant leaf disease. *Journal of Ambient Intelligence and Humanized Computing*, *14*(9), 12407–12424. https://doi.org/10.1007/s12652-022-04331-9

Kindt, R. (2020). WorldFlora: An R package for exact and fuzzy matching of plant names against the World Flora Online taxonomic backbone data. *Applications in Plant Sciences*, *8*(9), e11388.

Krishna, N., M, R., & Kaushik, R. (2020). Plant Species Identification Using Transfer Learning—PlantCLEF 2020. En L. Cappellato, C. Eickhoff, N. Ferro, & A. Névéol (Eds.), *Working Notes of CLEF 2020—Conference and Labs of the Evaluation Forum* (Vol. 2696). CEUR. https://ceur-ws.org/Vol-2696/#paper_139

Leiva Cuchia, M. A., & Rodriguez Mora, M. A. (2021). *Bogotá, diversa por naturaleza: Visibilización de la biodiversidad sinantrópica de la ciudad, a partir de material fotográfico, uso y diseño de sitios web. #biodiversidad rola.* http://repository.pedagogica.edu.co/handle/20.500.12209/16754

López Barrera, E. A., Fuentes Cotes, M. M., & Plata Rangel, Á. M. (2017). Percepción de actores involucrados en el estado de conservación del Humedal Torca-Guaymaral, Bogotá-Colombia. *Estudios Socioterritoriales*, *21*, 0–0.

Luis Alvarez, W. G. (2018). *Diagnóstico de la estructura ecológica principal de Bogotá durante los años 2008 – 2015*. https://hdl.handle.net/10654/20382

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, … Xiaoqiang Zheng. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. https://www.tensorflow.org/

Mersha, M., Lam, K., Wood, J., AlShami, A., & Kalita, J. (2024). Explainable Artificial Intelligence: A Survey of Needs, Techniques, Applications, and Future Direction. *Neurocomputing*, *599*, 128111. https://doi.org/10.1016/j.neucom.2024.128111

Meta Platforms. (2025). *React Native · Learn once, write anywhere*. https://reactnative.dev/

Michaels, T., Clark, M., Hoover, E., Irish, L., Smith, A., & Tepe, E. (2022, octubre 29). *Taxonomía de Plantas*. University of Minnesota.

Minhas, M. S. (2021, junio 5). Techniques for handling underfitting and overfitting in Machine Learning. *Towards Data Science*. https://towardsdatascience.com/techniques-for-handling-underfitting-and-overfitting-in-machine-learning-348daa2380b9/

Muñoz, I., & Bolt, A. (2021). *Diseño y desarrollo de aplicación móvil para la clasificación de flora nativa chilena utilizando redes neuronales convolucionales* (No. arXiv:2106.06592). arXiv. https://doi.org/10.48550/arXiv.2106.06592

Naveed, Q. N., Choudhary, H., Ahmad, N., Alqahtani, J., & Qahmash, A. I. (2023). Mobile Learning in Higher Education: A Systematic Literature Review. *Sustainability*, *15*(18), Article 18. https://doi.org/10.3390/su151813566

Pacheco-Prado, D., Bravo-López, E., & Ruiz, L. Á. (2023). Tree Species Identification in Urban Environments Using TensorFlow Lite and a Transfer Learning Approach. *Forests*, *14*(5), Article 5. https://doi.org/10.3390/f14051050

Pärtel, J., Pärtel, M., & Wäldchen, J. (2021). Plant image identification application demonstrates high accuracy in Northern Europe. *AoB PLANTS*, *13*(4), plab050. https://doi.org/10.1093/aobpla/plab050

Pérez Esteve, F. (2020). *Reconocimiento de imágenes de flora y fauna*. http://rua.ua.es/dspace/handle/10045/107799

Picek, L., Šulc, M., Patel, Y., & Matas, J. (2022). Plant recognition by AI: Deep neural nets, transformers, and kNN in deep embeddings. *Frontiers in Plant Science*, *13*, 787527. https://doi.org/10.3389/fpls.2022.787527

Quimbayo-Ruiz, G. A. (2016). Gestión integral de la biodiversidad en el Distrito Capital: Aportes para una gobernanza urbana. *Biodiversidad en la Práctica*, *1*. https://revistas.humboldt.org.co/index.php/BEP/article/view/47

Raz, L., & Zamora, H. A. (2023). *Catálogo de Plantas y Líquenes de Colombia* (Versión 1.3) [Dataset]. Universidad Nacional de Colombia. https://doi.org/10.15472/7avdhn

Rousavy, M. (2025). *Mrousavy/react-native-fast-tflite* [C++]. https://github.com/mrousavy/react-native-fast-tflite (Obra original publicada en 2023)

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2019). *MobileNetV2: Inverted Residuals and Linear Bottlenecks* (No. arXiv:1801.04381). arXiv. https://doi.org/10.48550/arXiv.1801.04381

Schrumpf, A., Killinger, M., Schiessle, P., & Scherp, A. (2024). On the coexistence of taxonomic botanical databases – a user study. *Willdenowia*, *53*(3), 309–316. https://doi.org/10.3372/wi.53.53308

Secretaría Distrital de Ambiente. (2022). *Actualización del Plan de Acción de la Política Pública para la Gestión de la Conservación de la Biodiversidad en el Distrito Capital 2022–2038* [Documento CONPES D.C.]. Alcaldía Mayor de Bogotá D.C. https://sdp.gov.co/sites/default/files/doc_conpes_dc_22_pp_biodiversidad.pdf

Secretaría Distrital de Planeación. (2021). *Documento de Diagnóstico POT 2022—2025*. Secretaría Distrital de Planeación. https://www.sdp.gov.co/sites/default/files/documento_diagnostico_dic2021.pdf

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2020). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *International Journal of Computer Vision*, *128*(2), 336–359. https://doi.org/10.1007/s11263-019-01228-7

Shapovalov, V. B., Shapovalov, Y. B., Bilyk, Z. I., Megalinska, A. P., & Muzyka, I. O. (2019). The Google Lens analyzing quality: An analysis of the possibility to use in the educational process. *Educational Dimension*, *1*, 219–234. https://doi.org/10.31812/educdim.v53i1.3844

SiB Colombia. (s/f). *¿Qué es el SiB Colombia?* Recuperado el 18 de agosto de 2025, de https://biodiversidad.co/acercade/sib-colombia/

Siddharth, T., Kirar, B. S., & Agrawal, D. K. (2022). *Plant Species Classification Using Transfer Learning by Pretrained Classifier VGG-19* (No. arXiv:2209.03076). arXiv. https://doi.org/10.48550/arXiv.2209.03076

Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., & Mueller, K.-R. (2021). *Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology* (No. arXiv:2003.05155). arXiv. https://doi.org/10.48550/arXiv.2003.05155

Tan, M., & Le, Q. V. (2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks* (No. arXiv:1905.11946). arXiv. https://doi.org/10.48550/arXiv.1905.11946

The DVC team and contributors. (2025). *DVC: Data Version Control—Git for Data & Models* [Software]. https://github.com/iterative/dvc

Visengeriyeva, L., Grigoriev, A., Kammer, A., Bär, I., Kniesz, A., Plöd, M., & Eberstaller, S. (s/f). *CRISP-ML(Q). The ML Lifecycle Process.* ML Ops: Machine Learning Operations. Recuperado el 14 de septiembre de 2025, de https://ml-ops.org/content/crisp-ml

Wagle, S. A., Harikrishnan, R., Ali, S. H. M., & Faseehuddin, M. (2021). Classification of Plant Leaves Using New Compact Convolutional Neural Network Models. *Plants*, *11*(1), 24. https://doi.org/10.3390/plants11010024

Wäldchen, J., & Mäder, P. (2018). Plant Species Identification Using Computer Vision Techniques: A Systematic Literature Review. *Archives of Computational Methods in Engineering*, *25*(2), 507–543. https://doi.org/10.1007/s11831-016-9206-z