UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Carrera de Ingeniería en Mecatrónica

Aplicación móvil para el reconocimiento de plantas medicinales mediante visión artificial

Trabajo de grado previo a la obtención del título de Ingeniero en Mecatrónica

Autor:

Carvajal Cuasquer Brayan Steven

Director:

Ing. David Alberto Ojeda Peña PhD

Ibarra - Ecuador

2023

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

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

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	0401881701		
APELLIDOS Y NOMBRES:	Carvajal Cuasquer Brayan Steven		
DIRECCIÓN:	Tabacundo – Tabacundo 2000		
EMAIL:	bscarvajalc@utn.edu.ec		
TELÉFONO FIJO:		**TELÉFONO MÓVIL:**	0986033569

DATOS DE LA OBRA	
TÍTULO:	Aplicación móvil para el reconocimiento de plantas medicinales mediante visón artificial.
AUTOR (ES):	Carvajal Cuasquer Brayan Steven
FECHA: DD/MM/AAAA	30/11/2023
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	**Ingeniería en Mecatrónica**
ASESOR /DIRECTOR:	**Ing. David Ojeda Peña, PhD / Ing. Iván García Santillán, PhD**

2. CONSTANCIAS

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

Ibarra, a los 30 días del mes de noviembre de 2023

EL AUTOR:

(Firma).....
Nombre: Carvajal Cuasquer Brayan Steven

Universidad Técnica del Norte

Facultad de Ingeniería en Ciencias Aplicadas

Constancia

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

Ibarra, 30 de noviembre de 2023

Carvajal Cuasquer Brayan Steven

C.I: 040188170-1

Universidad Técnica del Norte

Facultad de Ingeniería en Ciencias Aplicadas

CERTIFICACIÓN DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

En mi calidad de director del trabajo de grado "Aplicación móvil para el reconocimiento de plantas medicinales mediante visión artificial", presentado por el egresado Carvajal Cuasquer Brayan Steven, que opta por el título de ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, 30 de noviembre de 2023

Ing. David Alberto Ojeda Peña PhD

Director de Tesis

Universidad Técnica del Norte

Facultad de Ingeniería en Ciencias Aplicadas

APROBACIÓN DEL COMITÉ CALIFICADOR

El Tribunal Examinador del trabajo de titulación .Aplicación móvil para el reconocimiento de plantas medicinales mediante visión artificial.elaborado por Brayan Steven Carvajal Cuasquer, previo a la obtención del título de Ingeniero en Mecatrónica, aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

Ing. David Alberto Ojeda Peña PhD

Director de Tesis

Ing. Iván Danilo García Santillán PhD

Asesor de Tesis

Dedicatorias

Este trabajo de titulación está dedicado, primeramente, a las dos personas más importantes de mi vida. A mi madre, Elena Cuásquer, que gracias a sus deseos y bendiciones, esto fue posible. A mi padre, Miguel Carvajal, que sin sus consejos y ejemplos de superación, yo nunca hubiese podido ser una persona profesional, porque de él aprendí que ¡mil veces desanimado pero nunca rendido! A estas dos personas les agradezco con la vida por haberme dado la oportunidad de estudiar y por siempre haberme dado la fuerza necesaria para seguir.

A mi hermana, Angela Carvajal, quiero agradecerle por todos los chistes, las bromas y los buenos momentos de compañía cuando yo me sentía con angustia y desesperación a lo largo de mi carrera. Gracias porque siempre estuviste para sacarme sonrisas y darme motivos para seguir adelante. Espero que esto te sirva de ejemplo para hacerles sentir orgullosos a nuestros padres.

A usted, Carmita, también le dedico todo mi esfuerzo dejado en toda esta etapa universitaria. A usted, quien ha sido testigo de todo mi ánimo por ser alguien en la vida. Gracias por enseñarme que siempre hay que luchar para conseguir lo que uno quiere. Como usted dice, no es suerte, esto es sacrificio constante. Infinitas gracias a usted, cariño mío, por siempre estar aquí.

Para las cuatro personas que iniciaron conmigo pero que un día tuvieron que partir. Abuelitos, estén donde estén, lo hemos logrado. Me siento triste porque no están aquí conmigo en este logro, pero gracias por darme unos padres responsables.

Finalmente, todo este trabajo también está dedicado a Dios y a la Virgen de la Natividad. A la vez, les agradezco por darnos la fortaleza y la sabiduría para salir adelante a toda mi familia y amigos que estuvieron presentes en este largo proceso.

Agradecimientos

Quiero expresar mi más sincero agradecimiento al ingeniero David Ojeda Peña, tutor de este trabajo de titulación, por su sabiduría, paciencia y humildad al guiarme en el desarrollo y construcción de esta tésis. Su apoyo ha sido invaluable, y estoy profundamente agradecido por su dedicación y orientación durante todo el proceso.

De igual manera, quiero expresar mi agradecimiento a todos los docentes que integran la carrera de Ingeniería en Mecatrónica, por proporcionarme los conocimientos necesarios para mi desarrollo como profesional.

A la Universidad Técnica del Norte, específicamente a la Facultad de Ingenierías en Ciencias Aplicadas, le expreso mi más sincero agradecimiento por brindarme la oportunidad de formar parte de su distinguida institución y por permitirme llevar a cabo mis estudios de tercer nivel en este entorno académico.

Asimismo, quiero expresar mi más sincero agradecimiento al ingeniero Daniel Orbes, director ejecutivo de la empresa Ingenius Works, por brindarme la oportunidad de realizar mis prácticas pre-profesionales. Además, agradezco sus sabios consejos, los cuales han sido de invaluable contribución para la realización de este trabajo de titulación.

Índice general

Índice de figuras

Índice de tablas

Resumen

La visión artificial representa actualmente un gran desarrollo en la tecnología, siendo una rama fundamental de la inteligencia artificial (IA), puesto que abarca diversos campos de la investigación. Su influencia es tan significativa que se ha extendido incluso a actividades recreativas al aire libre, como caminatas, camping, trekking, entre otras. No obstante, en medio de estas actividades, persisten desafíos aún no resueltos. Un porcentaje significativo de la población involucrada no presta la debida atención al botiquín de primeros auxilios antes de iniciar dichas acciones, debido a que este elemento es quien debería proporcionarles los implementos y medicamentos necesarios para atender cualquier percance que pueda surgir en plena naturaleza.

Así pues, este trabajo de titulación presenta el desarrollo de una aplicación móvil enfocada a la identificación de cuatro plantas medicinales conocidas como: Manzanilla, Iso, Ñachag y Matuerzo; especies que crecen en la provincia de Imbabura. La metodología utilizada en este desarrollo se fundamenta en el modelo cascada, con el objetivo de mantener una organización rigurosa en la ejecución de las actividades programadas. Además, para determinar la solución óptima a la problemática planteada, se realiza un análisis basado en la ponderación de alternativas.

Las actividades realizadas se enfocaron en la creación de la base de datos, que abarcó un total de 4100 imágenes. De estas, 1100 fotografías correspondieron a la manzanilla, mientras que cada una de las otras especies tuvo 1000 representaciones. El entrenamiento de la red neuronal se llevó a cabo en Google Colab utilizando el modelo SSDMOBILENET V2, con la participación de las bibliotecas OpenCV y TensorFlow. En cuanto al diseño y construcción de la aplicación móvil, se empleó el entorno de desarrollo integrado Android Studio, utilizando el lenguaje de programación Kotlin.

Como resultado, se logró diseñar una aplicación móvil compatible con el sistema operativo Android, que realiza la inferencia con un modelo en formato .tflite para la identificación en tiempo real. Además, en la pantalla secundaria se proporcionan las propiedades medicinales, así como también el modo de uso, factores relevantes que cumplen la función de complementar al botiquín. El algoritmo de detección, evaluado mediante una matriz de confusión de forma general, demostró un rendimiento con una precisión del 93.65 %, lo que indica que las predicciones realizadas son confiables. Y de forma individual para cada variable se obtuvo los siguientes datos: Manzanilla, 98 %; Ñachag, 90 %; Iso, 94 % y Mastuerzo, 92 %. Cifras que demuestran que el modelo se sitúa sobre el 90 % manifestando que la identificación es precisa.

Palabras clave: Inteligencia Artificial, Visión Artificial, TensorFlow, OpenCv, Python, Google Colab, Kotlin, Android, Redes neuronales, Mobile net V2, Plantas medicinales, Matriz de confusión, Aplicación móvil,

Abstract

Machine vision currently represents a major development in technology, being a fundamental branch of artificial intelligence (AI), since it encompasses various fields of research. Its influence is so significant that it has even extended to outdoor recreational activities, such as hiking, camping, trekking, among others. However, in the midst of these activities, there are still unresolved challenges. A significant percentage of the population involved does not pay due attention to the first aid kit before starting such actions, since this element is the one that should provide them with the necessary implements and medicines to deal with any mishap that may arise in the middle of nature.

Thus, this degree work presents the development of a mobile application focused on the identification of four medicinal plants known as: Manzanilla, Iso, Ñachag and Matuerzo; species that grow in the province of Imbabura. The methodology used in this development is based on the cascade model, with the objective of maintaining a rigorous organization in the execution of the programmed activities. In addition, to determine the optimal solution to the problem posed, an analysis based on the weighting of alternatives was carried out. As a result, it was possible to design a mobile application compatible with the Android operating system, which performs the inference of a model in .tflite format for real-time identification. In addition, the secondary screen provides the medicinal properties, as well as the mode of use, relevant factors that fulfill the function of complementing the first aid kit. The detection algorithm, evaluated using a general shape confusion matrix, performed with an accuracy of 93.65 %, indicating that the predictions are reliable. predictions are reliable. And individually for each variable, the following data were obtained data: Manzanilla, 98 %; Ñachag, 90 %; Iso, 94 % and Mastuerzo, 92 %. Figures that demonstrate that the model is above 90 % showing that the identification is

accurate.

Keywords: Artificial Intelligence, Computer Vision, TensorFlow, OpenCv, Python, Goo-gle Colab, Kotlin, Android, Neural Networks, Mobile net V2, Medicinal Plants, Confusion Matrix, Mobile App confusion, Mobile application.

Introducción

Planteamiento del problema

Imbabura es un provincia mega diversa que contienen varias apreciaciones turísticas las cuales permiten desarrollar actividades como senderismo, camping, trekking, visitas de relajación, entre otras y a su vez, esta pluralidad también se ve comprendida en el área vegetal, puesto que existen distintos tipos de plantas medicinales en el entorno que pueden ser útiles para conservar la vida de los turistas en situaciones complicadas, demostrando así que el turismo y la medicina alternativa mantienen una alta relación que favorecen al desarrollo de la localidad.

Sin embargo, desde la perspectiva de la página web, reconocida como "ALMAOUTDOOR", especialista en actividades de aventura, manifiesta que: "El botiquín de primeros auxilios, es uno de los elementos más imprescindibles para los turistas y al mismo tiempo es uno de los más olvidados" [1] , debido a que los campistas no perciben el riesgo y el peligro que existen en las montañas del entorno, y tampoco cuentan con los conocimientos básicos sobre las propiedades de cada planta.

Por otra parte, las causas a las que están expuestos los turistas en cada actividad de aventura pueden ser: infecciones, traumatismos, picaduras o ataques de animales y insolación o hipotermia [2] , puesto que varios de ellos no toman en cuenta las condiciones climáticas para realizar dichas actividades.

Como resultado de los problemas mencionados anteriormente se propone en este trabajo de investigación abordar la problemática comprendida entre el olvido de los elementos de primeros auxilios, la falta de conocimiento de las propiedades curativas de las plantas medicinales y la exposición de los turistas a sufrir deshidrataciones por las fuertes temperaturas e hipotermia por las bajas temperaturas mediante el uso de visión artificial. Por lo que se plantea el desarrollo de un sistema capaz de identificar plantas medicinales en su ambiente natural, mostrando su respectivo nombre y a su vez la presentación de las propiedades y beneficios que pueden brindar.

Objetivos

Objetivo General

- Desarrollar un sistema identificador de plantas medicinales mediante visión artificial.

Objetivos Específicos

- Identificar las características de las plantas medicinales localizadas en el sector de Imbabura.

- Diseñar el sistema de visión artificial de acuerdo con los requerimientos obtenidos.
- Implementar el sistema de visión artificial para el reconocimiento de plantas medicinales.
- Validar el funcionamiento del algoritmo obtenido para la detección de plantas medicinales.

Alcance

El Presente trabajo de investigación está orientado a la creación de un sistema de visión artificial comprendido en la Deep learning o el aprendizaje profundo que configurará los parámetros básicos y entrenará a la aplicación móvil, por medio de un algoritmo desarrollado a base del lenguaje de programación Python y con librerías propias de este programa se pueda construir y adiestrar una red neuronal que logre detectar cuatro plantas medicinales dentro de su entorno natural y también brinde las propiedades como su modo de uso.
Por lo tanto, este trabajo iniciará con la obtención de imágenes dentro de su entorno natural y también con imágenes obtenidas por medio de una recopilación de diferentes páginas web situadas en la internet.

Seguidamente se procederá con el etiquetado del material gráfico a través de otra herramienta tecnológica que nos facilitará con la creación del dataset. Posteriormente, se trabajará en la validación de los etiquetados de cada imagen, para que al momento de la programación de la detección este no presente ningún problema y finalmente la exportación del módulo para continuar con el proceso de pruebas. Para la validación del código realizado en base a la detección

de plantas medicinales se efectuará una matriz de confusiones o también conocida como matriz de error en la cual evaluaremos el rendimiento de nuestro modelo entrenado.

Capítulo I

Revisión Bibliográfica

En esta sección se describen los cimientos teóricos para el desenvolvimiento del trabajo de titulación, así como también los aspectos más relevantes para la construcción del algoritmo de visión artificial, la detección de plantas medicinales y el diseño de la aplicación móvil.

1.1. Antecedentes.

Con el objetivo de detectar la parte foliar de una planta, se ha realizado el estudio enfocado en el aprendizaje profundo para la detección de hojas en las plantas (in situ) , donde se propone que el método más eficaz para la detección es la utilización de la red neuronal CNN con un total 1,4 millones de imágenes de las cuales el 60 % de imágenes se usaron para el entrenamiento y un 40 % para las respectivas pruebas; obteniendo como resultado para el conjunto de los datos de prueba una precisión del 96,1 %, mientras que para los datos del entrenamiento fue del 93,1 %. Estos resultados fueron obtenidos mediante pruebas realizadas con 100 imágenes de plantas capturadas durante el transcurso del día. Cabe mencionar que el sistema se tarda 0.5 segundos en realizar todo el proceso de detección [3].

Por otro lado, en una investigación enfocada en el reconocimiento de hojas de plantas medicinales a través de un algoritmo robusto extrajeron las características de textura, color y forma de las imágenes adquiridas y aplicaron diferentes tipos de redes neuronales artificiales para determinar cual de todas sería la más eficiente; obteniendo como resultados que el mejor modelo es la estructura 28-10-6. La precisión del modelo fue del 100 %, con un coeficiente de correlación y un error cuadrático medio de 1.00 y 2.35 x1012 [4].

En Base a la investigación sobre la detección profunda de objetos con modulación de predicción basada en atributos de ejemplo, utilizaron una modulación basada en atributos para modular el error de predicción; obteniendo como resultado que los objetos comunes en contexto demuestran que la EAPM (Módulo de protocolo de aplicación externa) aportan una mejora sustancial en los detectores de objetos profundos [5].

En cuanto a la informática y electrónica en la agricultura se realizó una detección de rasgos de fenotipado de plantas de tomate utilizando detectores de etapa única basado en YOLOv5, en donde los algoritmos de inteligencia artificial cumplen una función esencial en la automatización, estandarización y análisis de datos. Dentro del estudio utilizaron detectores de una sola etapa con el fin de detectar nudos, frutos y un conjunto de datos de múltiples genotipos de tomates. Encontrando como resultado, que los modelos logran puntajes altos considerando los desafíos de las imágenes de entrada en términos de tamaño de objeto, similitud entre objetos y su color [6] .

Con relación a aplicaciones móviles vinculados con la inteligencia artificial se desarrolló una aplicación móvil para detectar y clasificar enfermedades en las plantas de la uva con un modelo de detección de objetos basada en la red neuronal Faster R-CNN con la red troncal Inception-v2 para lograr una detección robusta y eficiente. De acuerdo con las pruebas realizadas con imágenes de enfermedades en la planta, se determinó que la precisión de la detección es de un 97,9 % mientras se ejecuta únicamente en un teléfono inteligente sin conectarse a un servidor [7].

La interacción de la inteligencia artificial con los teléfonos inteligentes cada vez es más ordinaria, puesto que se ha desarrollado una aplicación móvil para la detección de frutos de banano mediante la visión artificial. Este proceso empieza con la clasificación de fotografías para extraer las características principales de banano (borde y textura) y seguidamente se entrena el modelo mediante la red neuronal denominada CNN. Para el desarrollo de la aplicación se usó un editor de códigos en la nube y también la biblioteca llamada “Tensorflow”, con el cual se adaptó el modelo de la red neuronal en datos de entrada antes de implementarlo en una aplicación de Android. Como resultado se obtuvo que la aplicación tiene una precisión del 98,25 %, además reconoce al producto en tiempo real y brinda información del nivel de madurez al usuario. [8]

Los rasgos fenotípicos de las plantas pueden brindar información sobre las características fisiológicas de las estomas. Es por esta razón que el objetivo de esta investigación es detectar y reconocer de manera inteligente las estomas, a través del método de aprendizaje de las características y la red YOLOv4. Como resultados se ha obtenido que el método es superior a otros debido a que permite calibrar las medidas de detección, brinda resultados precisos y su costo

no es elevado [9].

En la revista científica AoB plants, realizaron un estudio comparativo entre las aplicaciones automatizadas para la identificación de plantas, en las cuales, se probaron 9 sitios web adecuados para usarse en el teléfono y un conjunto de 38 imágenes enfocadas en la flora vegetal superior para la detección. En este estudio se obtuvo como resultado un rendimiento mayor al 50 % y determinaron que la identificación es buena para las personas que recién empiezan en el área del reconocimiento , pero para los especialistas en botánica establecieron que aun necesita la validación de un experto en el área para analizar los resultados y continuar con los estudios ecológicos [10].

En base a la informática ecológica se realizó un análisis del rendimiento enfocado al reconocimiento de plantas en tiempo real con una red bilateral combinada. Pero reconocer en tiempo real y sin un entorno controlado es un verdadero desafío, debido a que el sistema esta expuesto a variaciones de iluminación, variaciones de escala, orientación del objeto y cambios de cámara. Dadas estas condiciones y con la aplicaciones de la red neuronal bilateral CNN, se obtuvo como resultado que el mejor modelo para estas aplicaciones es la combinación de MoDeNet + MLR, puesto que alcanzo una precisión del 99,39 % independientemente de los datos y el número de errores se redujo considerablemente para el reconocimiento de especies en tiempo real y sin ningún entorno controlado [11] .

Reconocer las plantas usando la forma exacta de ellas es una actividad complicada, puesto que el problema radica en lo dificultoso que es extraer con precisión las características de la hoja. Para ello se desarrollo un nuevo modelo denominado representación de distancia triangulo (TDR), en el cual, se muestran dos matrices: la primera matriz hace referencia a los signos, es decir, caracteriza la propiedad convexa y cóncava de la forma de la planta y la segunda se enfoca a la distancia que hay al centro del triángulo, en otras palabras al grado de inflexión que tiene el contorno Este modelo captura con eficacia las características de las plantas como también la forma de las hojas. Como resultados se obtuvo que de 4 conjuntos diferentes de hojas, el modelo reconoce con exactitud la forma de cada hoja y además es aplicaba para todas las plantas en general [12].

1.2. Plantas medicinales de Imbabura

En la provincia de Imbabura al norte del Ecuador, aun existen plantas medicinales que se conservan a pesar de la destrucción y la contaminación ocasionada por el ser humano. Es por ello, que en esta investigación se busca aprovechar las propiedades medicinales de cuatro plantas polinizadas de manera silvestre en el medio ambiente; de esta forma se busca asistir situaciones

de riesgo que pueden coaccionar ciertas actividades de recreación como caminatas, camping, trekking, entre otras. Además, este trabajo se basa en la detección a través de la visión artificial, en donde reconocerá a las flores de las siguientes plantas: la manzanilla, la ñachak, la pispura y el mastuerzo.

Para la elaboración de este proyecto se ha recopilado diferentes estudios en los que se demuestra las diferentes características sustanciales, las propiedades medicinales de cada una de las plantas mencionadas así como también su preparación y su modo de uso.

1.2.1. Manzanilla *(Matricaria recutita)*

La manzanilla es una importante planta medicinal y aromática que pertenece a la familia de las margaritas *(Asteraceae)* [13]. Sus flores se sitúan en la parte superior de las ramas y estas pueden ser de cabezuelas solitarias ó agrupadas que contienen un aroma inigualable . En si, la *Matricaria recutita*, no es tan exigente con la calidad del suelo y es adaptable a los diversos climas; de preferencia entre los 15 y 23°C [14].

Figura 1.1: Manzanilla *(Matricaria recutita)*

En cuanto a las propiedades medicinales se puede manifestar que gracias a la infusión de las hojas y las flores, esta logra una acción antiinflamatoria, espasmolítica, carminativa, antiulcerosa, digestiva, bactericida, fungicida y sedante suave, gracias a los diversos principios activos de la planta [15].

1.2.2. Ñachak *(Bidens andicola)*

La ñachak también conocida comúnmente como el amor ciego, mishico ó pante amarillo, ñachi, ñakachay es una planta medicinal que crece hasta unos 60 cm de altura con los tallos rectos, sus hojas son muy cortas y crecen en la parte superior del tallo. En cuanto a sus flores se agrupan en capítulos que crecen solitarios en el ápice de los tallos entre los meses de enero a marzo y son de un color amarillo encendido [16] como se puede visualizar en la figura 1.2.

Figura 1.2: Ñachag *(Bidens andicola)*

Las propiedades medicinales que tiene esta planta a través del uso de las flores son las siguientes.

- *Infusión.-* Ayuda a tratar la insolación por medio de paños húmedos, para afecciones a los riñones, y el hígado, la ictericia, afecciones hepáticas y para tratar golpes ó contusiones. Ingiriendo la infusión se puede aliviar el dolor de cabeza así como también el asma y facilitar el parto [17].
- *Machucada ó Desmenuzada.-* Ayuda a tratar escaldaduras e infecciones en los ojos [17].

1.2.3. Pispura ó Iso *(Dalea coerulea)*

La pispura ó también conocida como Iso, Izo, Alfilla del campo es una planta medicinal que tiene tallos de color rojizo con una longitud de de 8 a 16 cm. La hojas son semi carnosas, es decir que están dobladas a los largo de su nervio central y tiene glándulas en el envés que son apreciables como puntos oscuros. En cuanto a las flores que se puede visualizar en la figura 1.3 se encuentran al extremo del tallo con pétalos de colores variables entre azul, violeta y blanco. Florece entre los meces de marzo y abril. Crece en los suelos pedregosos, arenosos y el monte

[16].

Figura 1.3: Pispura *(Dalea coerulea)*

Las propiedades medicinales que tiene esta planta a través del uso de las hojas y flores son las siguientes.

- *Infusión.*- Sirve para limpiar la sangre y mitigarla neumonía, es decir, controlar el resfrío, anticatarrales, inflamaciones de las vías respiratorias y catarros [18].
- *Machucadas.*- Para controlar afecciones de la piel e infecciones parasitarias [19] .

1.2.4. Mastuerzo *(Tropaeolum majus)*

El mastuerzo o también conocida como capuchina es una planta medicinal de tallo delgado que en algunas veces permanece postrado y otras es un trepador por medio de los pecíolos foliares que actúan como zarcillo. Sus hojas son alternas con un cierto ángulo de desviación. En cuanto a sus flores como se puede visualizar en la figura 1.4 algunas son de color amarillo o anaranjado con una aproximación a rojo, estas presentan con un diámetro que va de 3 a 6 cm [20].

Las propiedades medicinales que tiene esta planta a través del uso de las hojas y flores son las siguientes.

- *Comestible.*- La planta es digestiva y aperitiva puesto que ayuda con la producción de jugos gástricos [21].

Figura 1.4: Mastuerzo *(Tropaeolum majus)*

- *Infusión.-* Por medio de su hojas y flores se puede controlar la inflamación de las vías urinarias y la hipertensión [21].
- *Machucadas.-* Por medio de las hojas y las flores machucadas se puede obtener beneficios anti-inflamatorios, además reduce el dolor muscular [21].
- *Otros aportes.-* Al ingerir esta planta se adquiere vitamina C, compuestos azufrados, es aperitiva y sirve para controlar enfermedades respiratorias [21].

1.2.5. Resumen de las propiedades medicinales y su uso

Entonces, con la información presentada sobre las propiedades medicinales, la preparación y el modo uso de las plantas; más la información recopilada en un estudio por la Facultad de Ciencias de la Salud de la Universidad Técnica del Norte, podemos resumir la información a través de la siguiente tabla .

Tabla 1.1: Propiedades medicinales.
[22]

Nombre Común	**Nombre Científico**	**Parte usada**	**Uso**	**Modo de Preparación**
Manzanilla	*Matricaria recutita*	Planta completa	1,2,3,4,7,8	Uso tópico, Infusión
Ñachag	*Bidens andicola*	Flores	1,3, 8, 9	Infusión
Pispura ó Iso	*Dalea coerulea*	Flores	1,2,3,5,6,7	Infusión
Mastuerzo	*Tropaeolum majus*	Planta completa	3,4,5,7	Uso tópico, Infusión

La tabla 1.1, muestra el nombre común, el nombre científico, la parte que se debe usar de la planta, el uso y el modo de preparación. En base al apartado donde se detalla por medio de números para que sirve cada una de las especies, esta hace referencia a cada uno de los siguientes ítem [22] :

- 1. Dolores (Articulaciones, cabeza, garganta).
- 2. Gastrointestinal.
- 3. Respiratorio.
- 4. Piel (Inflamación, golpes).
- 5. Renal-urológico.
- 6. Neurológico.
- 7. Anti-inflamatorio.
- 8. Fiebre.
- 9. Muscular.

1.3. Visión Artificial

La Visión Artificial o también conocida como visión por computador, es una disciplina compuesta por un grupo de habilidades que permiten la captura, el procesamiento y el análisis de imágenes con el principal objetivo de extraer información útil para dar respuesta a ciertas preguntas de acuerdo a la problemática planteada. Las técnicas necesarias para conseguir el objetivo provienen de dos áreas importantes las cuales son la ingeniería y la informática debido a que su naturaleza tecnológica a provocado que solo este al alcance de especialistas en la rama. Sin embargo, la aparición de herramientas, bibliotecas y librerías han echo que este trabajo reduzca su complejidad [23]. Y es aquí donde la visión artificial vinculada al aprendizaje automático hace que la actividad sea más exacta en la apreciación de los resultados [24].

En la figura 1.5 se puede visualizar que la visión por computador tiene una relación extensa con las demás áreas, y es por esta razón que esta área en conjunto con las otras es un ente con mayor relevancia, pero si nos enfocamos a la parte en estudio esta se ve comprendida por los siguiente subtemas.

1.3.1. Adquisición de imágenes

Mediante la adquisición de imágenes todos los escenarios presentes en el planeta dejan de ser luces y sombras sino que se traducen a información binaria, es decir, están compuestas por ceros y unos. Esta actividad es lo que usualmente se la realiza a través de una cámara [23] y

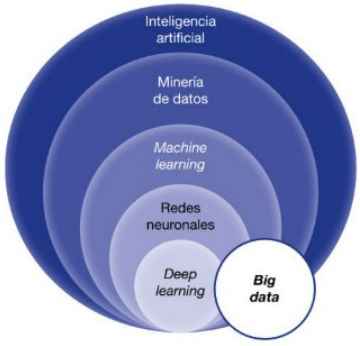

Figura 1.5: Relación de la Inteligencia Artificial con las otras área
[25].

a lo largo de la visión por computador y el aprendizaje profundo este apartado servirá para la creación de la base de datos.

Figura 1.6: Sistema para la adquisición de imágenes
[26].

Cabe mencionar, que para cumplir con la identificación de las plantas medicinales; la adquisición de imágenes se la realizará en su entorno natural, es decir, se hará la captura de fotografías en el medio ambiente.

1.3.2. Procesamiento y análisis de imágenes

En base al procesamiento, se utiliza diferentes algoritmos para resaltar las características de cada imagen. De esta forma se las prepara para usar nuevos algoritmos que permitan modificar las fotografías en cuanto a su color, tamaño y cambio de espacio. Por otro lado, el análisis de las

imágenes parte de los resultados de la fase anterior y en esta podemos realizar el reconocimiento, la clasificación y la identificación de lo que se haya entrenado previamente en el modelo esperado [23].

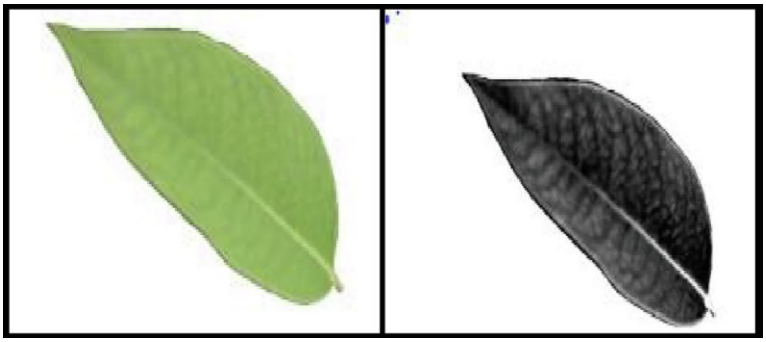

Figura 1.7: Procesamiento de imagen
[27].

1.3.3. Reconocimiento de imágenes

El reconocimiento de imágenes es la actividad de diferenciar objetos, personas, lugares o características en una fotografía o en un video que se han tomado hace un cierto tiempo o también en tiempo real [28] .

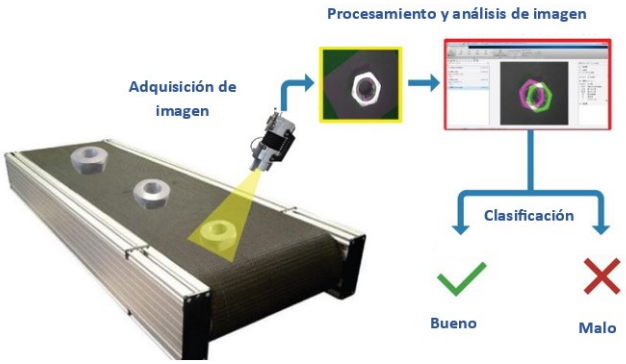

Figura 1.8: Reconocimiento de imágenes
[28].

Entonces, como se puede ver en la figura 1.8, el reconocimiento de imágenes es la parte más importante de todas las aplicaciones de la visión por computador, debido a que se identifica objetos o escenarios en imágenes, y a partir de esta información captada se toma decisiones para ayudar a otros sistemas de mayor complejidad [28].

1.4. Machine Learning.

Machine Learning (ML), es una ciencia que hace que los ordenadores memoricen por si solos [29], basándose en la construcción de algoritmos enfocados en la identificación de patrones y en el aprendizaje profundo para la predicción por medio de máquinas [30], enfocándose así en la adquisición de datos en tiempo real para aprender, adaptar y optimizar la salida de información [31]. Pero, qué es un algoritmo, y este es una serie de procedimientos que se ejecutan paso a paso, generalmente cálculos, que dan solución a un problema con un número finito de pasos [32].
De esta forma, una aplicación de ML orientada a la detección de plantas no va a discrepar de un programa clasificador de objetos; debido a que ambos se fundamentan en un algoritmo de Machine Learning que es capas de clasificar datos etiquetados [29].

Entonces, el proceso de aprendizaje profundo demanda de una gran cantidad de información con la que se busca brindar una respuesta a los datos de entrada. Cada entrada y respuesta del algoritmo significa un ejemplo más de aprendizaje y mientras más se obtenga, este optimizará mejor el modelo entrenado, de modo que logre determinar y predecir un resultado correcto aun cuando haya observado una entrada que no ha sido registrada. Estos resultados se deben a que las entradas y salidas se ajustan a una línea de dispersión estadística con la que se define el dominio del problema presentado [32].

Si los resultados arrojados se ajusta al dominio estadístico como se puede visualizar en la 1.9, es necesario afirmar que el modelo aprendió correctamente de las entradas proporcionadas [32].

1.4.1. Tipos de Machine Learning

Para que un ordenador pueda aprender por si solo es necesario que antes de todo el proceso del aprendizaje se deba identificar las tareas, conocer el entorno, las especificaciones a las cuales va a estar sometido el algoritmo y los problemas que va a solucionar, para que de esta forma se

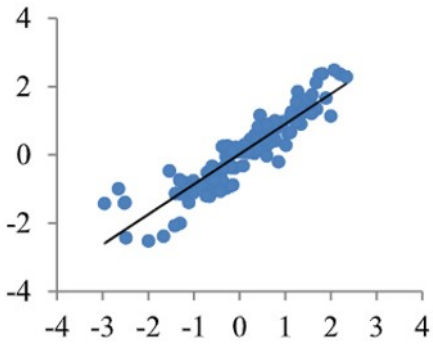

Figura 1.9: Gráfico de dispersión básico de datos de Machine Learning [33].

pueda clasificar el tipo de aprendizaje en uno de los siguientes ejemplares [29] y pueda trabajar de la mejor manera.

Tipos de aprendizaje

- Aprendizaje supervisado.

Este tipo de aprendizaje es aplicable cuando cada uno de los datos de entrada o también conocido como muestra tiene asociada una etiqueta. Es así como dentro de un conjunto de imágenes en la que cada una contenga un metadato se la puede etiquetar de la siguiente forma: (pict0001.bmp, "Manzanilla"), (pict0002.bmp, "mastuerzo"); y con la información obtenida a través de las etiquetas se puede utilizar un algoritmo de clasificación con el objetivo de entrenar un modelo y predecir la etiqueta guardada con una nueva imagen que no se encuentra en el conjunto de datos registrado para el adiestramiento del algoritmo [29].

Para nuestro caso de estudio, el etiquetado de imágenes se lo realiza por medio del software llamado *"LabelImg*, el cual nos permite etiquetar y registrar los metadatos encontrados en la fotografía. Para un mejor entendimiento de esto podemos revisar la Figura 1.10, donde se presenta la información que se desea registrar y el nombre respectivo de la etiqueta.

- Aprendizaje no supervisado.

El aprendizaje no supervisado ya no se cuenta con la respectivas etiquetas sino que este puede modificar las variables internas teniendo en cuenta la información de entrada. Es decir, en lugar de tener un conjunto de datos etiquetados el sistema de aprendizaje cambia los coeficientes del modelo basándose en el ambiente percibido y de esta forma desarrolla características de

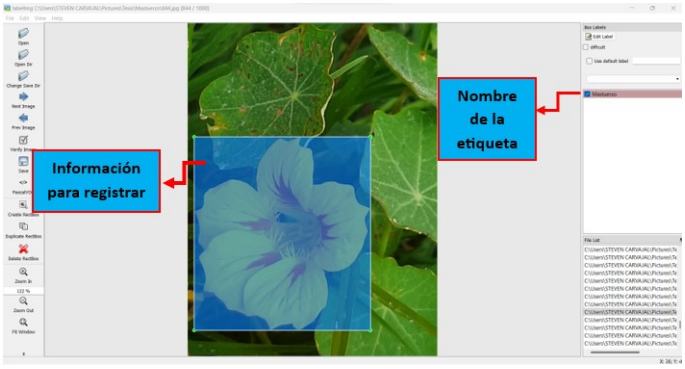

Figura 1.10: Etiquetado de imágenes

.

forma estadística para el universo de entrada [34]. En otras palabras, este tipo de sistemas son capaces de modificar los datos internos para adaptarse al entorno de una mejor manera[35].

La no supervisión se centra en que la red encuentre por sí sola características, regularidades, correlaciones o categorías en los datos de entrada, y se obtenga resultados de forma codificada a la salida. Estas respuestas son útiles siempre y cuando en los datos de entrada la información sea redundante; puesto que sin este factor para el modelo sería relativamente imposible encontrar patrones y solo se estaría procesando ruido a través del aprendizaje [35].

1.5. Red neuronal artificial.

Una red neuronal artificial es un tipo de algoritmo de machine ó deep learning que se fundamenta en sucesivas capas que trabajan en la transformación de datos; esta actividad realizada por los algoritmos tiene una gran semejanza a las conexiones neuronales del cerebro [25], debido a que este órgano es una red de elementos simples e interconectados que desempeñan actividades sobre la memoria, el pensamiento y la percepción [36].

Las RNA se enfocan en el aprendizaje de datos, siendo estos el resultado de combinar un conjunto de factores latentes no lineales y que, si somos capaces de aprender esta distribución, podremos predecir un nuevo conjunto de datos desconocidos con una cierta precisión. La arquitectura de la red neuronal básica esta compuesta por 3 capas principales en donde la primera será de entrada, la segunda será una capa oculta que representará el resultado de aplicar una transformación no lineal a los datos ingresados y una de salida. Los parámetros de una RNA

son los pesos de cada conexión que existen en la red y, a veces, un parámetro de sesgo . En la figura 1.11 podemos visualizar el esquema básico de una RNA, en donde la capa de entrada es de tercer tamaño, la capa oculta de cuarto y la salida es binaria [37].

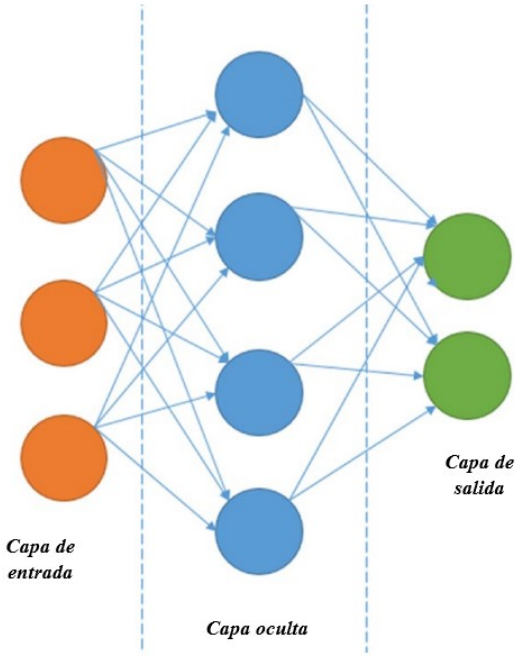

Figura 1.11: Red neurona artificial básica
. [37]

El procesos a seguir por la red neuronal para aprender cualquier tipo de entrenamiento de es mediante los siguientes pasos [37].

- *Definir la estructura o la arquitectura.*- Es la parte principal de una RNA, puesto que si se elige una red extensa con muchas neuronas ó también conocidas como unidades (en la figura 1.11, estas unidades son los círculos de cada etapa), podemos acoplar nuestros datos de entrenamiento y el modelo no se acoplará de manera correcta.

- *Elegir el tipo de transformación no lineal.*- Esta información será la que se aplica a cada conexión para controlar la actividad de la neurona.

- *Escoger la función de pérdida para cada capa.*- Esto solo se aplica cuando se tiene un aprendizaje supervisado, es decir cuando los datos se han etiquetado.

- *Aprender los parámetros de la red.*- Se determina los valores de cada peso de conexión, en la figura 1.11 estos pesos van situados en cada una de las flechas.

1.5.1. Red neuronal convolucional CNN

En referencia al funcionamiento que debe cumplir una red neuronal CNN, es aplicar procesos de convolución por medio de un filtro a cada imagen con distintas resoluciones de entrenamiento para identificar y captar las diferentes características como los bordes, el brillo, y de esta manera ir incrementando la complejidad de los rasgos que definen al objeto por medio de las decenas o cientos de capas inmersas dentro de ella, [28] en función de los patrones empleados. Para entender como funciona la extracción de características por medio de algoritmos es necesario revisar la figura 1.12, en donde se muestra que la extracción de datos de cada fotografía se la hace por medio de un análisis de los píxeles que presenta en su gráfico y de esta forma la red va recolectando las peculiaridades más representativas del objeto.

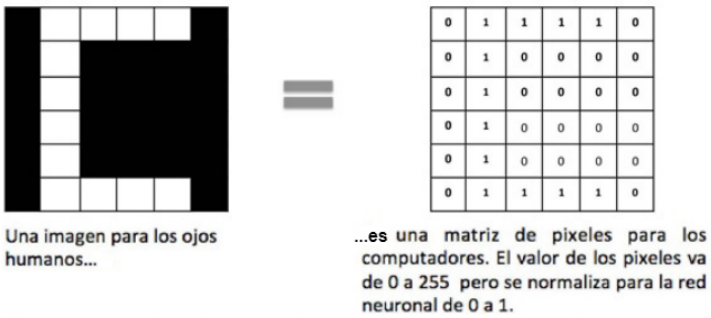

Figura 1.12: Extracción de características por medio de patrones [38].

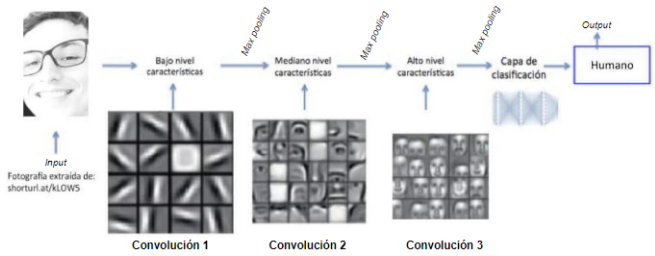

Figura 1.13: Arquitectura general y funcionamiento de una red neuronal CNN [39].

1.6. Base de datos

La recopilación de datos (Imágenes) es donde todo el procesos de machine learning comienza, aunque aparece como una actividad posterior a la identificación y definición del problema a resolver con ML. Después de haber identificado la problemática se ejecuta la recolección de toda la información necesaria mediante las cámaras de los dispositivos móviles o estáticos con la finalidad de captar los diferentes formatos, formas y tamaños de lo que se va a identificar [37], puesto que a partir de toda esta documentación y con la ayuda de diferentes programas se entrena el algoritmo que al final servirá para la detección. La base de datos se la puede presentar en el siguiente formato.

1.6.1. Archivo de datos CSV

Un archivo de datos CSV, es uno de los formatos más extensos, antiguos y preferido por los desarrolladores de ML debido a que los valores son separados por comas y son archivos que contienen datos con cada uno de los atributos que los caracteriza. La figura 1.14 presenta un archivo CSV básico de los datos con cada uno de los atributos delimitados por una coma ","[37].
Y es por este tipo de documentos que el algoritmo interpreta los datos para el respectivo entrenamiento de la red neuronal.

```
sno,fruit,color,price
1,apple,red,110.85
2,banana,yellow,50.12
3,mango,yellow,70.29
4,orange,orange,80.00
5,kiwi,green,150.00
6,pineapple,yellow,90.00
7,guava,green,20.00
```

Figura 1.14: Ejemplo de una base de datos CSV básica
. [37]

1.7. Herramienta para el etiquetado de imágenes.

Luego del proceso más importante en la visión artificial que es la formulación de la base de datos se continua por la anotación de la información que contiene cada una de las imágenes, y

esto se lo realiza por medio de herramientas que ayudan a registrar esta información. Existen diversas herramientas pero para el desarrollo de este proyecto se utiliza la herramienta digital llamada *LabelImg*.

1.7.1. LabelImg.

LabelImg es una herramienta que ayuda de forma rápida a realizar registros rectangulares en las imágenes que existan en la base de datos en formatos como PascalVOC y YOLO [**?**]. En si, esta aplicación es la que más cumple con los requisitos para el registro de los datos, puesto que tiene las siguientes características; primero brinda un soporte para múltiples clases de objetos, ayuda a cambiar el tamaño, interactúa inmediatamente con los archivos txt para el registro de las coordenadas y soporta secuencias de imágenes dentro de su entorno. A pesar de estos factores positivos la aplicación aun necesita un perfeccionamiento con la cual se pueda diferencia el color de las regiones, tener una vista paramétrica y un procedimiento óptimo para una mayor recopilación de fundamentos [40].

1.8. Biblioteca para la segmentación de imágenes

La segmentación de imágenes representa una cierta dificultad en la visión artificial. Y para ello se aplican herramientas que ayuden a dividir las imágenes en varios segmentos y objetos para su respectivo análisis [41]. A continuación se presenta una de varias bibliotecas que existen para realizar esta actividad.

1.8.1. OpenCV

Visión por ordenador de código abierto (Opencv), es una biblioteca gratuita creada por Gary Bradski en 1999, con la finalidad de revolucionar los procesos de la inteligencia y la visión artificial con una estructura sólida y eficiente [42]. Por otra parte esta desarrollada para trabajar dentro de la visión por ordenador, el aprendizaje automático, y el procesamiento de imágenes que ejercen una función esencial en las operaciones que se ejecutan en tiempo real[43]. Esta librería esta comprendida por herramientas, bibliotecas y módulos que incorporan soporte para implementar aplicaciones en las áreas antes ya mencionadas [44]. OpenCv se desarrolló en C y C++; y es ejecutable en Windows, Linux y Mac OS X. Además, brinda interfaces para Python, Java y Matlab como también bibliotecas transferibles a Android e iOS [42].

1.8.2. OpenCv frente a las imágenes.

Las imágenes son la información principal para una aplicación de visión artificial, puesto que ellas abarcarán la base de datos que se usará para la identificación de cualquier tipo de objeto. Entonces, es correcto afirmar que las fotografías son la entrada principal de una aplicación, donde la biblioteca OpenCV las interpreta en formato de matriz para poder almacenar los detalles de las imágenes, como la anchura, altura, profundidad, canal y otros [44].

Esta acción dentro de Python la realiza la biblioteca NumPY ya que contiene varios algoritmos numéricos y operaciones que permiten trabajar a través de matrices y arrays de gran tamaño [44].

1.9. Biblioteca para la detección.

La acción de ver para la mayoría de los seres humanos es una actividad sencilla, pero detrás de esta labor existen millones de células visuales y neuronas que interactúan para capturar la luz, decodificar las señales y procesarlas en niveles altos de abstracción [45]. Lo que acabamos de mencionar es lo mismo que hace un sistema de detección y por ende podemos apreciar una cierta semejanza, debido a que un sistema de ML de acuerdo a la base de datos obtenido a través de las etiquetas generadas es capas de detectar.

1.9.1. Tersorflow

Tensorflow es un ecosistema desarrollado y lanzado en el 2015 por el equipo de google Brain [46], y esta mantiene un software flexible y escalable para cálculos numéricos a través de gráficos con un cierto flujo de datos. Esta librería junto con las herramientas que ofrece permite a los desarrolladores programar y entrenar de forma eficaz redes neuronales y otros ejemplares enfocados en el aprendizaje automático. Los algoritmos que usa tensorflow están construidos por medio del lenguaje C++ y están automatizados por la arquitectura de dispositivos unificados también conocida por sus siglas como CUDA, una plataforma de cálculos paralela al tema de estudio creada por NVIDIA. Python, también ofrece en su ambiente la facilidad de usar en sus entornos esta biblioteca, puesto que es uno de los lenguajes más completos y estables a comparación de los otros [47].

En la figura 1.15 es posible visualizar como el modelo entrenado detecta y brinda información sobre lo que se esta enfocando en ese momento y esto se obtiene gracias al trabajo en conjunto de la base de datos con la respectiva biblioteca para la detección.

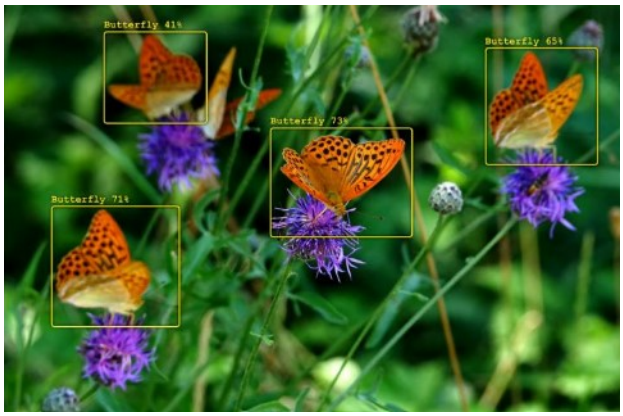

Figura 1.15: Detección de objetos mediante el uso de bibliotecas.
[45]

1.10. Distribuidor de lenguajes de programación - Anaconda

Anaconda es un distribuidor de código abierto que brinda lenguajes como Python y R, con el objetivo de simplificar la instalación, la gestión y el despliegue de paquetes de diferentes extensiones, es decir este sistema brinda todas las herramientas en un solo programa. Además, permite usar de manera fácil más de 1500 librerías y paquetes direccionadas al tratamiento de datos. En cuanto a los ordenadores es compatible con Windows, Linux O Mac y dentro de su distribución contiene entornos de desarrollo integrado como Jypter y Spider [48] que facilitan el desarrollo de programas básicos y complejos dentro del desarrollo de la programación.

1.10.1. Python

Python, es un lenguaje de programación de alto nivel, diseñado para crear tareas complejas. Además, posee una sintaxis fácil de leer y limpia, lo cual permite la construcción rápida de códigos, con menos errores y una mayor facilidad para mantener y actualizar el mismo [49]. Es decir, es un estilo que se desarrolló tanto para aprender como para programar, debido a que sus comandos no son difíciles de entender y están orientados a diseñar aplicaciones con una escritura sencilla pero con una capacidad de baja y alta jerarquía. Además soporta paradigmas de la programación como la orientada objetos y tiene un sistema de gestión de memoria de forma dinámica brindando así amplías y extensas bibliotecas [50] que permiten realizar más actividades en conjunto con otras áreas de la informática como lo es la visión por computador.

1.10.2. Entorno de desarrollo integrado - Jupyter

Jupyter es un entorno de programación interactivo que se basa en la integración de la consola. Además incorpora procesos de desarrollo y ejecución de códigos con la implementación de documentos que se encuentran en el mismo contexto. Este ambiente en sus inicios se diseñó a partir de un entorno llamado Interactive Python como también contenía ideas del propietario de Mathematica. Ahora Jupyter es uno de los más conocidos puesto que su uso se ha centrado en la docencia y la investigación en especial en la robótica y la ciencia de datos [51].

1.11. Aplicaciones Móviles

Las aplicaciones móviles también son conocidas como apps y es un software que a estado presentes entre la humanidad desde hace un gran tiempo, tanto así que en los primeros teléfonos que desarrollo Nokia especialmente el modelo 1100 ya estaban incluidas como son los calendarios, las alarmas y las calculadores. Ahora en la actualidad estas ya existen de diversos tipos, formas y colores. Con la aparición de los teléfonos inteligentes las apps tuvieron una mayor acogida y posteriormente se convertirían en un ente rentable para la sociedad [52].

1.11.1. Sistema operativo android

El sistema operativo que contienen los dispositivos móviles brindan características importantes para el desarrollo de aplicaciones. Android es el sistema operativos más usado en los teléfonos inteligentes y tablets puesto que brinda adaptaciones para diferentes dispositivos de los distintos fabricantes. Fue diseñado por Open Handset con una alianza de más de 80 empresas, entre ellos Google es el personaje principal [53].

1.11.2. OpenCv para Android

Para la segmentación de imágenes que se la realizará es necesario que la biblioteca de OpenCv sea compatible con android. Es decir android debe admitir el acceso a sus funciones por medio de su aplicación nativa y también por su Api de contenedores de Java. Para el caso de usar los contenedores de Java de OpenCv su código directamente importará las bibliotecas nativas de OpenCv utilizando JNI. Y esta elección depende del programador, sin embargo si hace uso de las llamadas nativas generar menos gastos generales de JNI pero requiere más esfuerzo de programación. Por otra parte si usa Java implica menos esfuerzo de programación pero mas recursos de JNI [54]

Capítulo II

Metodología

En este capítulo se puntualiza el enfoque, los tipos de investigación a los cuales está orientado este trabajo, así como también el análisis y la metodología aplicada para el desarrollo del algoritmo de visión artificial, y la construcción de una aplicación móvil cuya finalidad será la detección de plantas medicinales dentro de su entorno. Adicionalmente, se describen cada una de las actividades contempladas para cumplir con los objetivos específicos propuestos.

2.1. Enfoque de investigación / Tipo de investigación.

Este trabajo está referido a una investigación aplicada, debido a que busca generar conocimiento en base a una práctica encaminada a los problemas presentes en la sociedad [55] . Adicionalmente, se lleva a cabo una investigación documental, que facilitará la recopilación de información bibliográfica en base a las plantas medicinales que existen y que se usan en la provincia de Imbabura. Así mismo, se aplica una investigación de campo, puesto que, se ejecutará una compilación de datos esenciales en un entorno natural [56], es decir, se efectuará la recopilación de imágenes de cada planta en su propio ambiente para la creación de la base de datos.

2.2. Diseño de Investigación.

A continuación, se exponen las etapas y actividades secuenciales para la realización de este proyecto. La figura 2.1 muestra un resumen de las labores que se deben cumplir con base a los objetivos específicos de este trabajo; seguidamente se hace un amplio análisis de cada una de las herramientas tecnológicas que conllevarán al desarrollo de este trabajo de titulación.

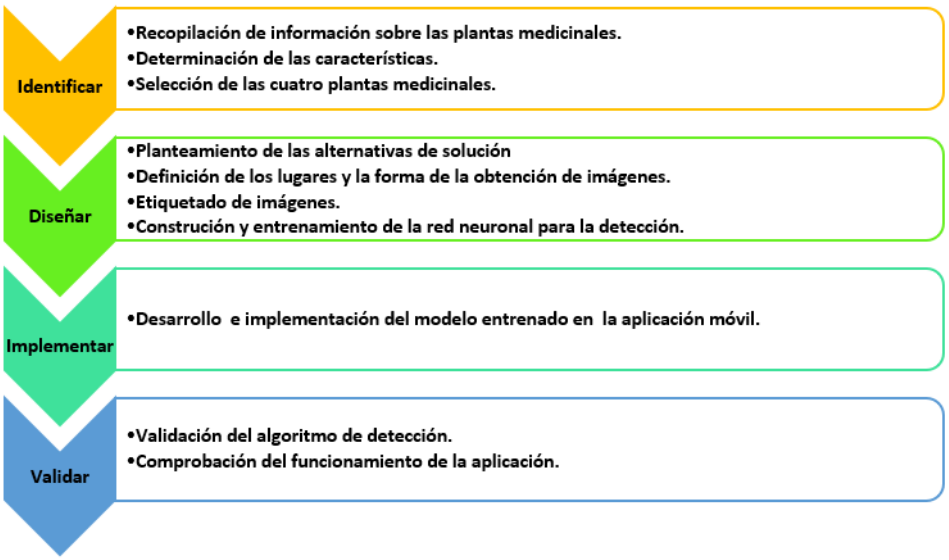

Figura 2.1: Cronología de actividades de acuerdo a los objetivos específicos.

2.3. Cronología de actividades

El desarrollo de este trabajo de titulación esta guiado por medio de los objetivos específicos, en donde cada objetivo representa una etapa y esta contienen diferentes actividades que se encuentran sometidas bajo una metodología conocida como modelo cascada. Esto se debe a que nos permite definir y estructurar todas las fases, teniendo en cuenta que los requerimientos, el diseño, el desarrollo, las evaluaciones, la puesta en funcionamiento y el mantenimiento son factores importantes que deben considerarse a lo largo de todo el proceso de desarrollo [57]. Este modelo presenta rigidez en cuanto al avance del proyecto, debido a que cada fase depende obligadamente del paso anterior , es decir, mientras no se haya concluido un proceso no es posible avanzar al siguiente. Las actividades necesarias se presentan a continuación.

2.3.1. Etapa 1

- Identificar las características de las plantas medicinales localizadas en el sector de Imbabura.

En esta actividad se verifican los estudios realizados en la provincia. Para ello se revisan los repositorios bibliográficos pertenecientes a la Universidad Técnica del Norte, específicamente adscritos a la Facultad de Ingeniería en Ciencias Agrícolas y Ambientales. Adicionalmente, se buscan trabajos e investigaciones a través de las bases de datos científicas, tales como Science-

direct, Taylor and Francis, IEEE, entre otras. Finalmente, se lleva a cabo una entrevista a una persona especializada en la medicina ancestral y certificada por el Ministerio de Salud Pública (*MSP*). Además, es nativa de la provincia y posee conocimientos sobre las plantas medicinales localizadas en la zona.

Actividad 1.1: **Determinación de las características físicas de las plantas medicinales.**

A partir de la entrevista realizada, se logró identificar las plantas medicinales utilizadas en la provincia. Estos resultados están disponibles en el capítulo 1, sección 1.2, donde se describen en detalle las propiedades, características, la parte de la planta a emplear y el modo de uso. Dado que la identificación de plantas por medio de la visión artificial puede resultar algo complejo, se lleva a cabo una entrevista interdisciplinaria con el director ejecutivo de la empresa *Ingenious Works*, quien se especializa en el desarrollo e implementación de la visión artificial. Esto se realiza con el propósito de examinar las características fenotípicas de cada especie, tales como el color, forma de la flor y el tamaño de la planta, para determinar cuáles son viables para la detección.

Actividad 1.2: **Selección de plantas medicinales a identificar según el análisis respectivo.**

De acuerdo con la información obtenida en la actividad 1.1 , las plantas seleccionadas son las que presentan flores con: formas, texturas y colores únicos dentro del entorno natural. Esto implica que las características de las mismas deben ser visualmente llamativas, es decir, que sus colores sean fácilmente distinguibles en medio de la naturaleza. En relación a la información recopilada durante la entrevista a la persona con experiencia en la medicina ancestral y al análisis de las características realizada por el experto en visión artificial, se ha identificado las siguientes plantas como las más pertinentes, considerando sus propiedades medicinales y las necesidades del proyecto: Manzanilla, Ñachag, Iso y Mastuerzo.

2.3.2. Etapa 2

- Diseñar el sistema de visión artificial de acuerdo con los requerimientos obtenidos.

De acuerdo al diagrama de flujo presentado en la figura 2.2. En esta segunda etapa se plantea el desarrollo del algoritmo de visión artificial, en base a los criterios obtenidos sobre las características de las plantas en el punto 2.3.1.

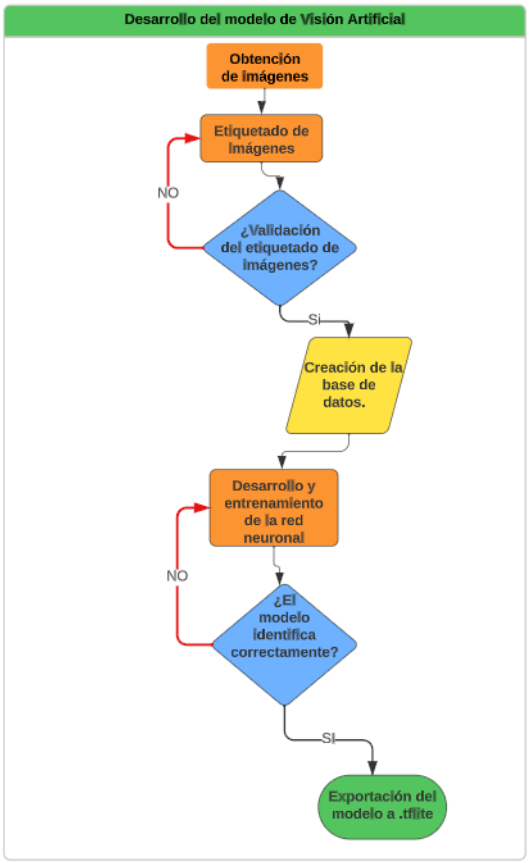

Figura 2.2: Diagrama de flujo para el desarrollo del entrenamiento de la red neuronal.

Como se puede visualizar en el diagrama de flujo, la primera actividad es la obtención de imágenes, pero para poder cumplir con todas las actividades pertinentes para el objetivo específico número 2 es necesario la aplicación de una matriz morfológica, con la cual se pueda determinar los instrumentos y herramientas necesarias para el desarrollo de las actividades. Esto se lo realiza con finalidad de agilizar los procesos para evitar inconvenientes con las actividades que conllevan al entrenamiento de la red neuronal utilizada.

Matriz Morfológica

La problemática planteada tiene varias formas de resolverse, se hace uso de la matriz morfológica con la finalidad de examinar diferentes alternativas y elegir la solución más viable en referencia a las limitaciones identificadas. A su vez, también nos permite relacionar varias herramientas tecnológicas con diversos componentes que cumplen la misma función pero que aveces son incompatibles con los otros componentes que se están utilizando. De esta forma se

puede elegir la trayectoria con los recursos que mejor se adaptan a la solución buscada.

En la figura 2.3 se presenta la matriz morfológica con tres posibles alternativas que son analizadas a continuación.

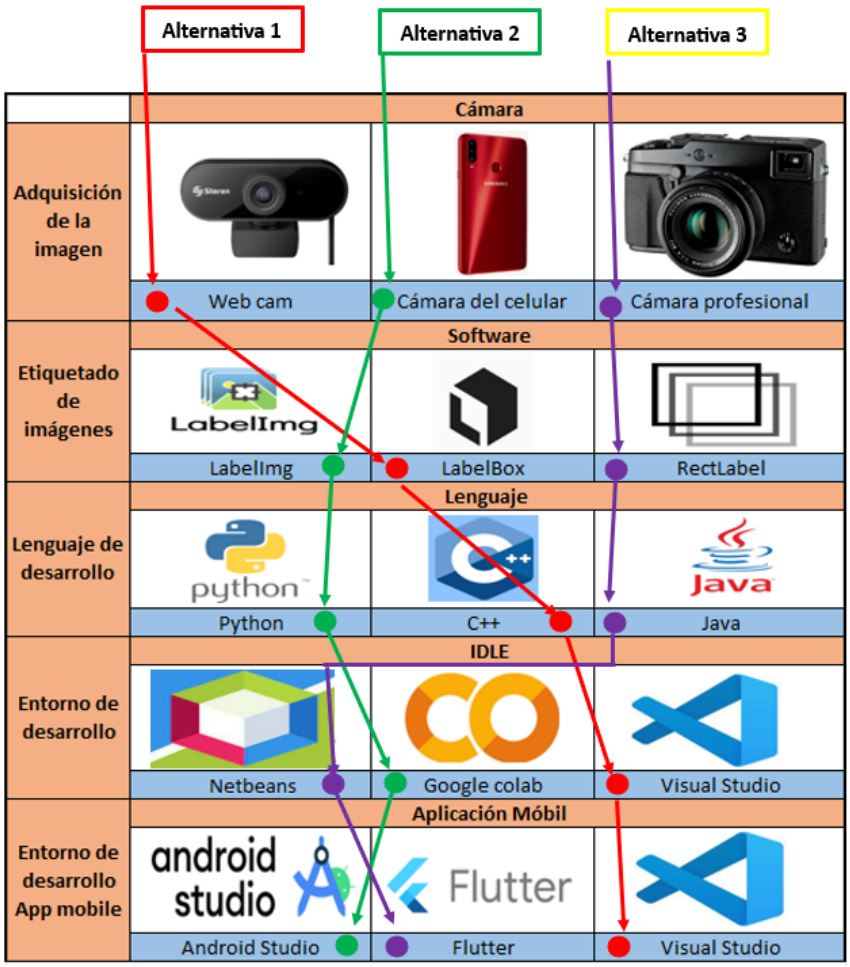

Figura 2.3: Matriz morfológica

- *Alternativa 1.-* Como parte de la alternativa uno se pretende adquirir las imágenes por medio de una webcam, la cual es un dispositivo que no brinda las mejores condiciones para capturar fotografías al aire libre, puesto que esta diseñada para trabajar en un ambiente controlado. Para el etiquetado de datos se trabajará con LabelBox, un software que es fácil de trabajar y además, para el desarrollo, el entrenamiento de la red neuronal y el diseño de la aplicación se aspira utilizar el lenguaje C++ y el entorno de desarrollo denominado Visual Studio.

- *Alternativa 2.-* En la alternativa dos las imágenes van a ser tomadas por medio de la cámara de un teléfono inteligente, que es un dispositivo que trabaja con toda normalidad al aire libre. El etiquetado se lo va hacer a través del software LabelImg, que es un programa que registra las coordenadas del objeto a identificarse en archivos de diferentes formatos. La red neuronal va ser desarrollada por medio del lenguaje de python y se va a usar Google Colab como medio de desarrollo, puesto que este entorno se encuentra situado en la nube y además aprovecha los recursos del internet. Finalmente la aplicación será desarrollada en Android Studio, un programa creado para el desarrollo de Apps móviles.

- *Alternativa 3.-* En esta alternativa las fotografías van a ser capturadas por medio de una cámara profesional, la cual trabaja muy bien en todos los ambientes, tiene muy buena resolución pero su costo es elevado. El registro de los datos es mediante un software conocido como RectLabel que permite etiquetar objetos en imágenes y vídeos. Por otro lado el desarrollo y entrenamiento de la red neuronal se lo va a realizar por medio del lenguaje de Java en el entorno conocido como Netbeans. Para concluir la aplicación será diseñada en Flutter.

Evaluación de alternativas.

Dado que en la matriz morfológica se han presentado tres alternativas que brindan una solución a la problemática planteada, es necesario relacionar estas opciones con las especificaciones del sistema con la finalidad de determinar cual de las tres posibilidades es la que mejor se ajusta y cumple con los criterios para la solución esperada.

Ponderación de criterios.

De acuerdo a las necesidades presentadas, a las investigaciones y observaciones realizadas en otros estudios, los criterios que se deben cumplir con el sistema de detección por medio de la visión artificial son los siguientes:

1. Imagen
2. Procesamiento
3. Precisión
4. Velocidad
5. Trabajo (Similitud a las condiciones de trabajo)

En relación a los criterios definidos, en la tabla 2.1 se realiza la ponderación de cada uno de ellos en cuanto al nivel de importancia que tienen respecto a los requerimientos necesarios, en donde la mayor puntuación es 1 y el puntaje más bajo es 0. Es necesario resaltar que los criterios “imagenz “trabajo", son los que que más relación tienen con los otros, dado que el porcentaje de ponderación es de 0.24 %, valor que debe ser tomado en cuenta para el desarrollo del sistema de visión artificial, puesto que mantiene una alta vinculación con los demás términos.

Tabla 2.1: Ponderación de criterios.

	Imagen	**Procesamiento**	**Precisión**	**Velocidad**	**Trabajo**	$\sum+1$	**Ponderación**
Imagen	-	1	1	1	1	5	0.24
Procesamiento	1	-	0.5	1	1	4.5	0.22
Precisión	1	0.5	-	0	1	3.5	0.16
Velocidad	0	0.5	0.5	-	1	3	0.14
Trabajo	1	1	1	1	-	5	0.24
Total	-	-	-	-	-	21	1

Ponderación de alternativas.

Debido a la importancia que tienen los criterios del sistema y las tres posibles soluciones planteadas en la matriz morfológica, se hace uso de la ponderación de alternativas para relacionar cada uno de los parámetros con la finalidad de comparar y evaluar las diferentes opciones antes de elegir una alternativa que resuelva la problemática encontrada.

Por lo tanto, en la tabla 2.2 se analiza la relación que presenta el criterio imagen ante las posibles soluciones.

Tabla 2.2: Ponderación de alternativas para el criterio Imagen.

Imagen	*Alternativa 1*	*Alternativa 2*	*Alternativa 3*	$\sum+1$	*Ponderación*
Alternativa 1	-	0	0	1	0.15
Alternativa 2	1	-	0.5	2.5	0.39
Alternativa 3	1	1	-	3	0.46
Total	-	-	-	6.5	1

Como se puede visualizar los resultados en la tabla 2.2, la alternativa que cumple con el criterio es la número 3.

A continuación, en la tabla 2.3 se examina el vínculo que existe entre el criterio de procesamiento en base a las alternativas de solución.

Tabla 2.3: Ponderación de alternativas para el criterio Procesamiento.

Procesamiento	*Alternativa 1*	*Alternativa 2*	*Alternativa 3*	$\sum+1$	*Ponderación*
Alternativa 1	-	0	0	1	0.15
Alternativa 2	1	-	1	3	0.46
Alternativa 3	1	0.5	-	2.5	0.39
Total	-	-	-	6.5	1

De acuerdo al análisis realizado en la tabla 2.3, la alternativa que cumple con el criterio de procesamiento es la número tres, dado que el porcentaje de 0.46 % es mayor a los otros.

Seguidamente en la tabla 2.4, las alternativas que brindan una mejor precisión son la 2 y la 3, puesto que el porcentaje de ponderación es igual al 0.50 %. Debido a este resultado donde se obtiene dos alternativas de solución, es necesario aclarar que al finalizar la ponderación de alternativas se realizará una tabla la cual estará compuesta de todos los resultados obtenidos y se elegirá la opción correcta de acuerdo a los datos.

Tabla 2.4: Ponderación de alternativas para el criterio de Precisión.

Precisión	*Alternativa 1*	*Alternativa 2*	*Alternativa 3*	$\sum+1$	*Ponderación*
Alternativa 1	-	0	0	0	0
Alternativa 2	1	-	0.5	2.5	0.5
Alternativa 3	1	0.5	-	2.5	0.5
Total	-	-	-	5	1

En la tabla 2.5 se estudia la importancia que existe entre el criterio de velocidad (tiempo que se demora el sistema en identificar una planta) y las 3 alternativas propuestas.

Tabla 2.5: Ponderación de alternativas para el criterio de Velocidad.

Velocidad	*Alternativa 1*	*Alternativa 2*	*Alternativa 3*	$\sum+1$	*Ponderación*
Alternativa 1	-	0.5	0	1.5	0.25
Alternativa 2	1	-	0.5	2.5	0.42
Alternativa 3	1	0	-	2	0.33
Total	-	-	-	6	1

Se obtiene como resultado que la alternativa con mayor importancia para el criterio de velocidad es la opción número 3, debido a que en el análisis a obtenido un porcentaje del 0.42 % más que las otras.

Para finalizar la ponderación de las alternativas en referencia a los criterios planteados, a continuación se examina la relación que tiene cada una frente al criterio de trabajo, puesto que es un aspecto fundamental para el método de visión artificial, debido a que aquí el sistema es tratado como si estuviera trabajando de forma natural, es decir, en un entorno situado en el medio ambiente en la cual un individuo cuenta con los recursos necesarios y óptimos para la situación que esta experimentando en ese momento.

Tabla 2.6: Ponderación de alternativas para el criterio de Trabajo.

Trabajo	*Alternativa 1*	*Alternativa 2*	*Alternativa 3*	$\sum+1$	*Ponderación*
Alternativa 1	-	0	0	1	0.20
Alternativa 2	1	-	1	3	0.60
Alternativa 3	0	0	-	1	0.20
Total	-	-	-	5	1

Entonces, con los resultados que se ha obtenido en la tabla 2.6 es posible manifestar que la alternativa que cumple y además cuenta con los requisitos para que una persona trabaje de manera tranquila en el entorno es la alternativa tres, ya que ha obtenido un 0.60 % del total.

Después de haber analizado cada uno de los criterios de forma separada en base a las posibles soluciones, ahora podemos agrupar toda esta información en una sola tabla y verificar cual es la mejor opción para el desarrollo de este trabajo de titulación.

Tabla 2.7: Ponderación de alternativas.

	Imagen	**Procesamiento**	**Precisión**	**Velocidad**	**Trabajo**	$\sum$	**Ponderación**
Alternativa 1	0.24x0.15	0.22x0.15	0.16x0	0.14x0.25	0.24x0.20	0.15	3
Alternativa 2	0.24x0.39	0.22x0.46	0.16x0.5	0.14x0.42	0.24x0.60	0.48	1
Alternativa 3	0.24x0.46	0.22x0.39	0.16x0.5	0.14x0.33	0.24x0.20	0.37	2
Total	-	-	-	-	-	1	*Alternativa 2*

En vista a la evaluación realizada donde se expone a las tres alternativas planteadas frente a los criterios seleccionados, la mejor opción para dar cumplimiento a los objetivos específicos y a los criterios es el ítem número 2, puesto que en la tabla 2.7 se puede evidenciar que tiene como ponderación el primer lugar y además cuenta con un porcentaje de 0.48 %, siendo este valor mayor a los demás.

Puesto que, el uso de la cámara del teléfono no es perjudicial tanto para el desarrollo como para el entrenamiento de la red neuronal. Si realizamos una comparación con los otros dos componentes, es posible manifestar que la web cam no es costosa pero no brinda buenos resultados

a la intemperie, en cambio el costo de la cámara profesional es muy elevado pero sus resultados son notorios en cuanto a la calidad de la imagen, sin embargo es muy influyente en el entrenamiento de la red neuronal. También se debe recalcar que la aplicación móvil para identificación de las plantas medicinales va estar expuesta a ambientes no controlados, donde la luz del día es un factor que se debe tomar en cuenta y las imágenes van ser captadas por medio de celulares, lo que demuestra que la mejor opción y la que más se acerca a las condiciones de trabajo es la alternativa 2.

Después de haber determinado la alternativa que da solución a este trabajo de titulación es posible continuar con el desarrollo de las actividades. A continuación se presenta los pasos que se debe seguir para dar cumplimiento con el objetivo especifico mencionado.

Actividad 2.1: **Obtención de las imágenes para la creación de la base de datos.**

Para la creación de la base de datos, primero se verifica la ubicación de las plantas seleccionadas, lo que nos condujo a las siguientes ubicaciones:

Tabla 2.8: Localización de las plantas medicinales.

Provincia	**Cantón**	**Especie encontrada**
Pichincha	Pedro Moncayo	Todas las especies
Imbabura	Ibarra	Todas las especies
Imbabura	Otavalo	Todas las especies
Carchi	Espejo	Todas las especies

Dado que estos lugares presentados en la tabla 2.8 son muy frecuentados por el autor de este trabajo de titulación, se realiza la captura de las fotografías por medio de un dispositivo móvil; en particular, un teléfono inteligente de la marca Samsung, modelo A20s, el cual posee triple cámara trasera de 13 Mpx , 5 Mpx y 8 Mpx dedicada a mejorar la resolución de cada imagen. La ñachag, la pispura y el mastuerzo son especies compuestas por un total de 1000 fotografías cada una, mientras que la manzanilla consta 1100 fotos en su carpeta de datos. Esta diferencia en la cantidad de imágenes se debe primeras 100 imágenes de la última planta fueron obtenidas de la internet. Es importante puntualizar que la base de datos cuenta con fotografías capturadas desde distintos ángulos con la finalidad de recopilar la mayor cantidad de información a través de ellas.

En la figura 2.4, se evidencia la creación de las cuatro carpetas que contienen las imágenes especificadas anteriormente.

Figura 2.4: Base de datos para las especies: Iso, Manzanilla, Mastuerzo, Ñachag.

Actividad 2.2: **Etiquetado de imágenes por medio del software *LabelImg*.**

Dentro del análisis de alternativas por medio de ponderaciones, se determinó que la mejor solución para el etiquetado de imágenes es el software *LabelImg*. La primera actividad de esta fase consiste en descargar el programa desde el repositorio Github. Para ello, se indaga en las diferentes cuentas registradas en el sistema y se verifica cual de todas proporciona el programa libre de virus. Como resultado de esta investigación, se determina que el perfil conocido como *EdjeElectronics* gestionado por un ingeniero experto en hardware y especialista en el campo de la visión artificial, brinda esta herramienta como parte de su proyecto titulado *TensorFlow Object Detection API Tutorial Train Multiple Objects Windows 10*. A continuación se adjunta el enlace a su repositorio: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10. Y los pasos a seguir para el etiquetado de las imágenes son los siguientes:

1. Debido a que la página es fácil de comprender, el único paso es descargar el código en formato ZIP. Para el desarrollo de este trabajo, se descargó y descomprimió en el disco local *C*, dentro de una carpeta llamada tensroflow.

2. Dado que, al descomprimir, se obtiene dos carpetas con los nombres *"LabelImg"* y *"models"*, se ha visto la necesidad de utilizar un entorno de desarrollo integrado denominado *Anaconda* . Para ello, procedemos a descargarlo de la página oficial cuyo enlace es el siguiente: https://www.anaconda.com/download. Dentro del enlace la única tarea que debemos realizar es seleccionar el sistema operativo del ordenador que se esta utilizando e instalarlo. Una vez finalizada la instalación, la interfaz principal que se visualiza es la que se presenta en la figura 2.5. Además, esta IDE, nos ofrece la facilidad de crear nuevos entornos de trabajo con diversas bibliotecas y herramientas de aplicación para el desarrollo de ciertos códigos de programación.

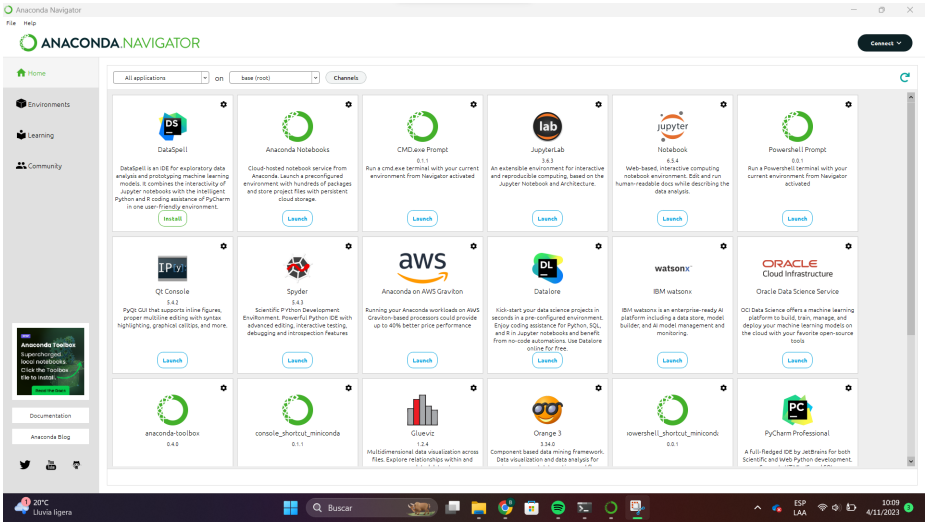

Figura 2.5: Interfaz principal del entorno de desarrollo integrado *Anaconda*.

3. Una vez que el ordenador cuenta con el entorno de desarrollo, vamos abrir el Símbolo del sistema o también conocido por sus siglas *cmd* a través del buscador de Windows.Una vez en el *cmd* navegaremos hasta la carpeta de tensorflow1 > labelImg y finalmente solicitaremos que inicie la aplicación mediante código como se muestra en la figura 2.6. Si, al digitar la última linea, el programa no se inicia, es posible que falten algunas bibliotecas. En ese caso, puede instalar las siguientes librería por medio de los comandos de programación: pip install pyqt y pip install lxml, uno a la vez. Luego de esto ya puede ejecutar el inicio del programa.

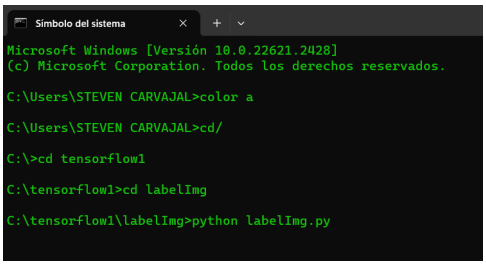

Figura 2.6: Inicio de la aplicación por medio de código

La interfaz principal del programa se muestra en la figura 2.7. En esta pantalla, existen iconos que permiten cargar la carpeta que contiene todas las imágenes (dataset) , seleccionar la ubicación de destino donde para guardar los archivos en formato txt, y permite modificar el formato de los archivos. En nuestro caso, trabajamos con el formato *Pascal-Voc*. Además, hay un icono que permite delimitar la planta, introducir su etiqueta y esta será visible en el cuadro de etiquetas, en ingles denominado como box labels.

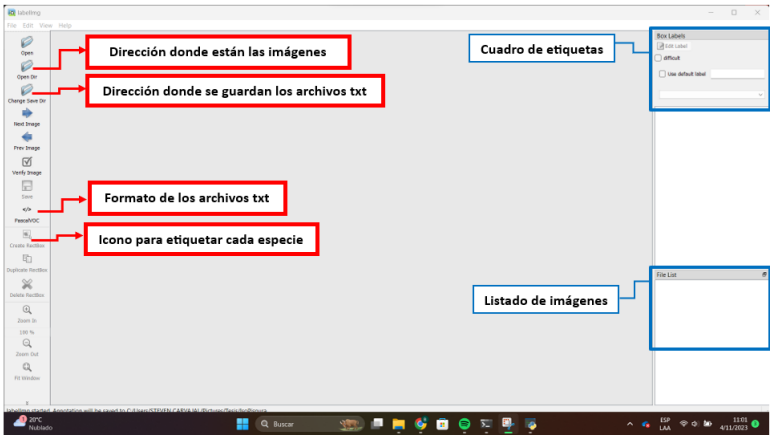

Figura 2.7: Software *LabelImg*

Para concluir, después de cargar las imágenes, seleccionar la ubicación para guardar y el formato de los archivos txt, ya es factible etiquetar todas las fotografías de la base de datos. Cada vez que se etiquete una nueva imagen, automáticamente el archivo txt se generará y se guardará en la ruta especificada. El la figura 2.8, se puede apreciar el proceso mencionado anteriormente.

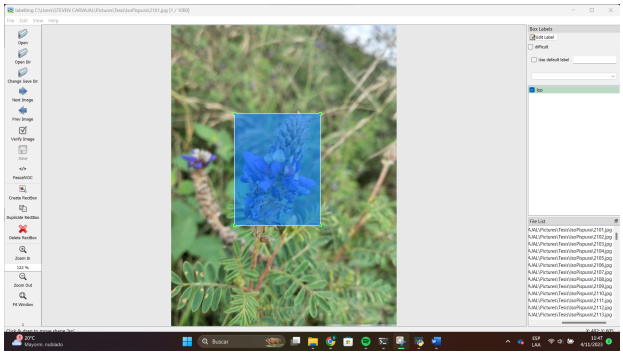

Figura 2.8: Etiquetado de imágenes en el Software *LabelImg*

4. Es crucial revisar la carpeta donde se encuentran los archivos txt después de etiquetar los datos. En ocasiones, pueden surgir inconvenientes en los que estos archivos no se generan y, por consecuencia no aparecen en la ubicación especificada. Esto podría ocasionar problemas durante el entrenamiento de la red neuronal, puesto que la cantidad de imágenes no coincidiría con la cantidad de archivos y no solo eso, en un paso posterior la base de datos se divide en el 80 % de la información para el entrenamiento y el 20 % para la validación y aquí puede surgir problemas graves, dado que no sabemos en que parte dichos archivos no fueron encontrados.

Actividad 2.3: **Desarrollo y entrenamiento del modelo para la detección de plantas.**

Para obtener el modelo de identificación en formato *.tflite*, que es el que se necesita para trabajar con el diseño de la aplicación móvil se siguieron los siguientes procesos.

1. Primero, lo que se necesita es una cuenta que disponga de google drive, con la única finalidad de cargar la base de datos en ella.

2. Seguidamente, en el buscador de google vamos a escribir *google colab*, y entrar a la primera opción que arroje la búsqueda en la internet. Esto nos lleva a un entorno de desarrollo de códigos a través del lenguaje de python alojado en la nube, con la única finalidad de emplear los recursos proporcionados por ella y no utilizar las prestaciones de la computadora, debido a que el tiempo de entrenamiento puede variar si el equipo no tiene buenas características. Una vez que nos encontramos aquí, vamos a proceder con la creación de un nuevo cuaderno, para el caso descrito se denomina *Plant Detection.ipynb*, como se puede evidenciar en la figura 2.9.

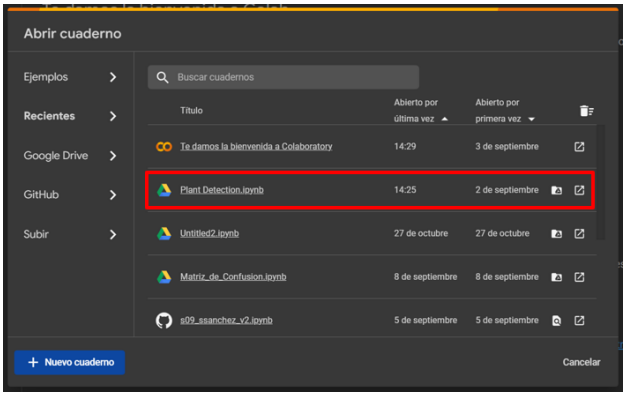

Figura 2.9: Cuaderno de desarrollo en *Google Colab*.

3. El siguiente paso implica configurar el entorno de desarrollo. Para esto, es necesario dirigirse a la opción *editar* de la barra de tareas. En el menú desplegable, seleccionaremos *configuración del cuaderno*. Esto abrirá una nueva pantalla que permite elegir el tipo de entorno de ejecución y el acelerador por hardware. En este caso, se debe elegir GPU T4, lo que proporcionará un mejor rendimiento en el aprendizaje profundo de la red neuronal. Este proceso se evidencia en la figura 2.10.

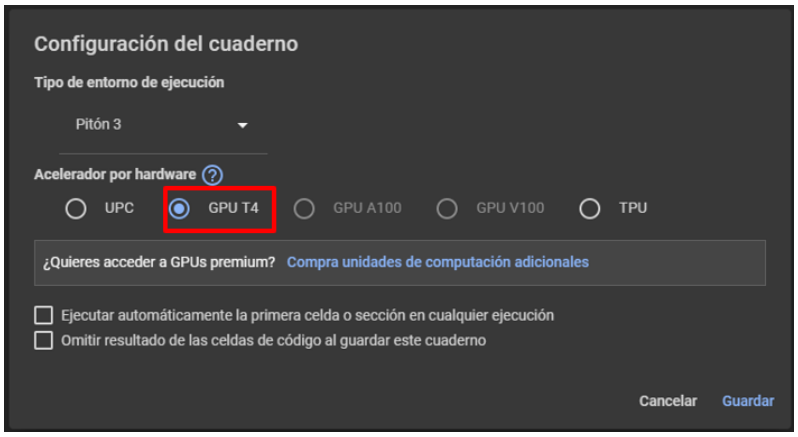

Figura 2.10: Configuración del entorno de ejecución.

4. Ahora, basándonos al desarrollo del código de programación que finaliza con el entrenamiento del modelo, lo primero que se debe hacer es establecer la comunicación del entorno de ejecución con Google Drive. Esto se puede evidenciar en el primer y segundo bloque de códigos de la figura 2.11. Luego, se crea una carpeta principal llamada *modelo entrenar* la cual contendrá dos subcarpetas denominadas *datos* y *entrenamiento*.

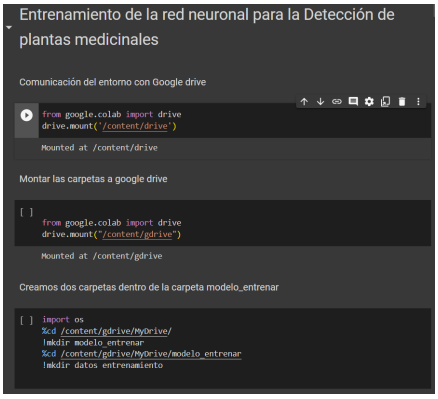

Figura 2.11: Comunicación del entorno de ejecución con Google Drive.

5. El paso siguiente constituye en abrir la carpeta de *datos*, creada en el paso anterior dentro de Google Drive. Luego, cargar las carpetas comprimidas en formato .ZIP de las imágenes y de las anotaciones obtenidas al momento de etiquetar las fotografías, es decir, los archivos txt. Además de estos elementos, se debe incluir un nuevo documento denominado *labelmap.txt*, en el cual se registrarán a las etiquetas de las especies a identificar, en consecuencia , manzanilla, Nichag, Iso y mastuerzo. Estas variables deben ser escritas en diferentes lineas.

6. Continuando con el procedimiento, en el entorno de programación, vamos a instalar las librearías necesarias para la detección de objetos. Además, llevamos a cabo la instalación de la API de TensorFlow, la cual la obtendremos directamente desde el repositorio oficial del servidor, así como también la API de detección. Estos pasos se visualizan en la figura 2.12. Debido a las frecuentes actualizaciones de las bibliotecas, si surgen inconvenientes al momento de instalar las librerías o durante el desarrollo del código debido a las diferencias entre las versiones, se recomienda consultar la página oficial del servidor. A continuación, se adjunta el enlace correspondiente *https://www.tensorflow.org/?hl=es-419*.

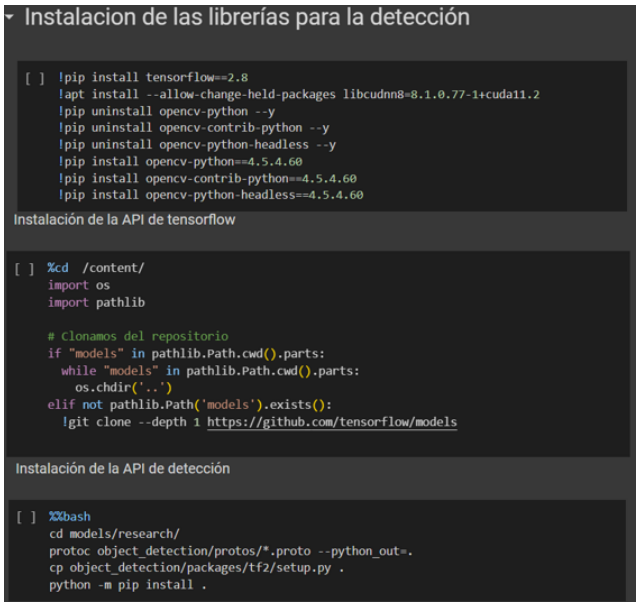

Figura 2.12: Librerías y APIs de detección.

7. Ahora procedemos a extraer los archivos de las carpetas comprimidas, esto se lo puede revisar en el primer bloque de programación de la figura 2.13 . Seguidamente creamos

dos carpetas; la primera se denomina *etiquetas train*, donde se guardarán el 80 % de las etiquetas para llevar a cabo el entrenamiento; y la segunda se la llamará *etiquetas validation*, donde se almacenará el 20 % de las etiquetas que, al concluir la actividad, se utilizará como método de evaluación del modelo entrenado. Para finalizar este ítem, el último bloque realiza el cálculo del 20 % de 4100 archivos, obteniendo como resultado 820 documentos que serán tomados en cuenta para la validación. En la figura 2.14, se muestra la programación para extraer el valor calculado de archivos de forma aleatoria. Con el último bloque de programación de la figura, se crean los archivos cvs, en el cual se detalla la dirección de las imágenes y las coordenadas del objeto etiquetado.

Figura 2.13: División de las etiquetas de la base datos, 80 % para el entrenamiento y 20 % para la validación.

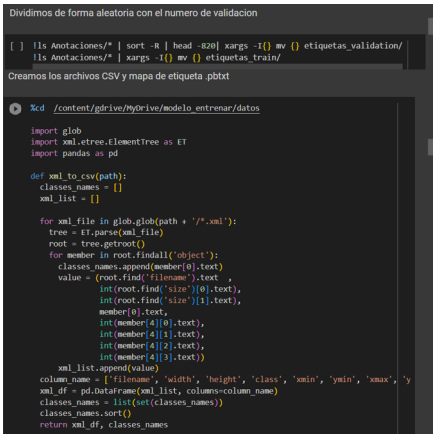

Figura 2.14: Creación de los archivos en formato *.cvs* con las etiquetas de la base de datos.

8. Para finalizar con el preprocesamiento de los datos, es necesario crear los archivos tfrecord.py. Estos documentos tienen la función de almacenar registros binarios, es decir, son elementos complementarios que sirven para el entrenamiento. En la figura 2.15 , se presenta los códigos necesarios.

Figura 2.15: Creación de los archivos tfrecord.py.

9. Ahora para iniciar con la etapa del entrenamiento, lo primero que se debe hace es crear una carpeta denominada *premodelo* para almacenar el modelo pre entrenado que va a ser descargado desde el enlace adjunto: *https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md*, esta actividad se la puede evidenciar en la primer bloque de código de la figura tal 2.16. La arquitectura de la red neuronal a usarse es SSD de Mobilenet en su segunda versión, puesto que la primera versión esta en el proceso de ser obsoleta y la versión más reciente aun esta en constantes actualizaciones, el código que permite la descarga de esta red se la evidencia en el segundo bloque de código de la figura 2.16

Figura 2.16: Descarga del modelo SSD Mobilenet V2 pre entrenado.

10. Para continuar con el proceso del entrenamiento, es necesario configurar el archivo *pipeline_nuevo_config*. En primer lugar, en la línea 3, se especifica la cantidad de clases a identificar, que en este caso son 4. Luego, se ajusta el tamaño del lote de entrenamiento, también conocido como bach size, en ingles. Este valor depende del total de imágenes y del tamaño de la memoria RAM de la GPU que contenga el entorno. En el caso de estudio, se utiliza un valor de 35.
    En la linea 165, se establece la ubicación del modelo pre-entrenado y se elimina todo lo que este después de ckpt-0. Finalmente, en las lineas 175 y 185, se coloca la dirección del archivo label_map.pbtxt. En la línea 177 la dirección del archivo train.record, y en la línea 189, la ubicación del documento validation.record. El archivo esta en la sección de anexos.

11. El proceso de entrenamiento se inicia con el código que se muestra en la figura 2.17. En el caso propuesto, se utilizan un total de 60000 pasos o también conocidos como steps. Es importante tener en cuenta que el tiempo de ejecución puede variar según el rendimiento del equipo, la cantidad de imágenes y la arquitectura del modelo a entrenar.

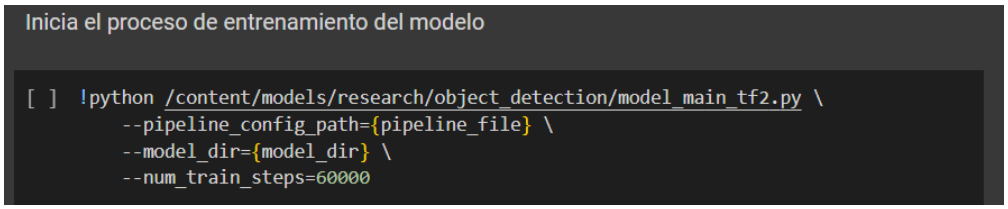

Figura 2.17: Codificación para empezar el proceso de entrenamiento.

    Durante esta actividad, por medio de la consola se puede evidenciar los pasos transcurridos, las perdidas encontradas y el rango de aprendizaje. Cabe recalcar que cada 10000 pasos se guarda un documento en la carpeta checkpoint, Este procesos tuvo una duración de 2 días.

12. Para finalizar el entrenamiento , se crea una carpeta donde se almacenará el modelo entrenado en formato .pb. Seguidamente se verifica la ubicación donde se guardará el archivo y se ajusta la configuración del mismo, todo ello basado en el código que se muestra en la figura 2.18.

Figura 2.18: Exportación del modelo en formato .pb del modelo de identificación de plantas.

Actividad 2.4: **Exportación del modelo de TensorFlow a TensorFlow lite ;**

Para poder usar el modelo entrenado dentro de una aplicación móvil, es necesario convertir el archivo a un formato conocido como TensorFlow Lite. Este documento final en formato .tflite contiene la información necesaria para procesar los datos de entrada, realizar la identificación de la planta y mostrar una respuesta, ya sea identificando una especie o no. Pero esto se lo logra por medio de los siguientes pasos.

1. Para poder continuar con el proceso de conversión, es necesario de de una librería que nos permita esta transformación, para lo cual en un nuevo bloque de código se debe digitar la siguiente linea: *!pip install tf-nightly*, después de esto es necesario reiniciar el entorno de ejecución.

2. El siguiente paso consiste en exportar el modelo utilizando el archivo *export_tflite_grapth _tf2.py*. Ademas se debe verificar las ubicaciones de los archivos necesarios, para ello el bloque de código mostrado en la figura 2.19 es el necesario. A veces surgen inconvenientes en las rutas de destino, para lo cual es recomendable dar click derecho, copiar la dirección de la carpeta y pegarla en el código.

Figura 2.19: Exportación del modelo al formato .tflite.

3. Después de haber convertido el modelo, se procede a guardarlo en una ruta especifica. Este elemento por defecto se guarda en punto flotante 32, pero aun es posible convertirlo a punto flotante 16 con el siguiente código.

```
[ ]  import tensorflow as tf
     import numpy as np


     saved_model_dir = '/content/gdrive/MyDrive/modelo_entrenar/datos/tflite/saved_model
     dataset_dir = '/content/gdrive/MyDrive/modelo_entrenar/datos/tflite'

     converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
     converter.optimizations = [tf.lite.Optimize.DEFAULT]
     converter.target_spec.supported_types = [tf.float16]
     tflite_quant_model = converter.convert()

     # save model
     open("/content/gdrive/MyDrive/modelo_entrenar/datos/tflite/model_fp16.tflite", "wb"
```

Figura 2.20: Exportación del modelo a punto flotante 16.

4. La siguiente actividad es agregar los metadatos para darle una correcta funcionalidad a la aplicación móvil, por consecuencia es necesario instalar la siguiente librería *!pip install tflite_support_nightly*.

5. Para finalizar el proceso de conversión, es necesario el archivo labelmap.txt, donde se encuentran escritas las etiquetas de cada especie. Seguidamente es necesario verificar que las direcciones de cada carpeta sean las correctas, debido a que de ahí se va a extraer los metadatos y es donde se va a guardar el modelo. La figura 2.21 representa la información necesaria del código.

   Es importante recalcar que el código completo, con los documentos necesarios para el entrenamiento de la red neuronal y la transformación al formato de TensorFlow Lite se encuentra disponible en el repositorio de Github, para ello se comparte el enlace a la pagina del autor de este trabajo de titulación, https://github.com/SpedyGo/Object-detection-with-TensorFlow. Si con el transcurso del tiempo se encuentran inconvenientes con la incompatibilidad de las versiones de cada biblioteca se recomienda visitar la pagina oficial de Tensorflow.

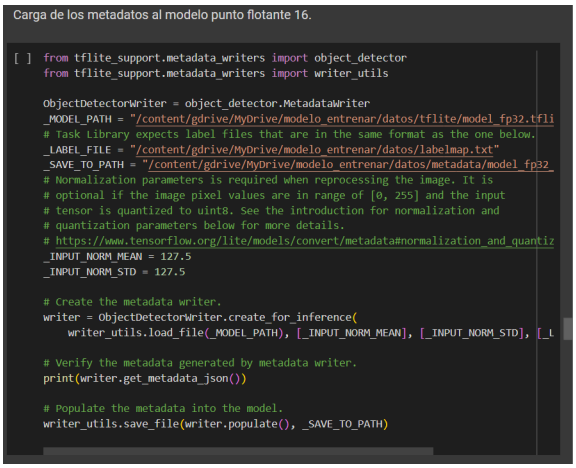

Figura 2.21: Inclusión de los metadatos en el modelo TensorFlow Lite

2.3.3. Etapa 3

- Implementar el sistema de visión artificial para el reconocimiento de plantas medicinales.

Para desarrollar una aplicación móvil con la inferencia de un modelo en formato *.tflite* existen diversas plataformas disponibles, y una de las más reconocidas es Android Studio, puesto que proporciona las mejores herramientas para el diseño de aplicaciones. Dicho esto, a continuación se detallan las subactividades que conducen a obtener como resultado final una aplicación para la detección de plantas medicinales a través de la visión artificial.

Actividad 3.1: **Desarrollo de la aplicación móvil en Android Studio;**

1. El primer paso implica la instalación del entorno de desarrollo integrado de Android Studio. El software se lo puede descargar desde la página oficial del servidor a través del siguiente enlace: *https://developer.android.com/studio?hl=es-419*. Al acceder, seleccionen el botón *Descargar Android Studio Giraffe*, esto los dirigirá a nueva pestaña en donde tendrán que aceptar los términos y condiciones para habilitar la opción de descargar del software.

2. El proceso de instalación comienza al darle click derecho y ejecutar como administrador el instalador descargado. Esto abrirá una nueva pantalla en la que se debe seleccionar *next*, a continuación se verificará la ruta donde se va a guardar el programa y, finalmente, se ejecuta la instalación. El tiempo de instalación puede variar de acuerdo a la capacidad

del equipo. Al abrir el software, se puede configurar de manera estándar o personalizada, dependiendo de las preferencias de cada programador.

3. Dado que la biblioteca de TensorFlow es una librería de código abierto y brinda ejemplos de aplicaciones sencillas que permiten cargar el modelo entrenados en formato *.tflite*, se procederá a descargar un modelo compatible con Android. Esto requiere dirigirse al repositorio oficial de TensorFlow en Github por medio del siguiente enlace: *https://github.com/tensorflow/examples/tree/master/lite/examples/image_classification/android* y descargar la carpeta mencionada.

4. El siguiente paso es abrir el modelo con Android Studio. Para hacerlo, debes seleccionar la opción *open* y navegar hasta donde se encuentra la carpeta del proyecto. En este caso de estudio, el archivo se encuentra en el escritorio. La figura 2.22 muestra el proceso a seguir. Si el sistema del programa muestra un advertencia sugiriendo la actualización del Gradle de la aplicación a una versión más reciente, no dude en seleccionar "si", puesto que es posible que necesite ciertas actualizaciones.

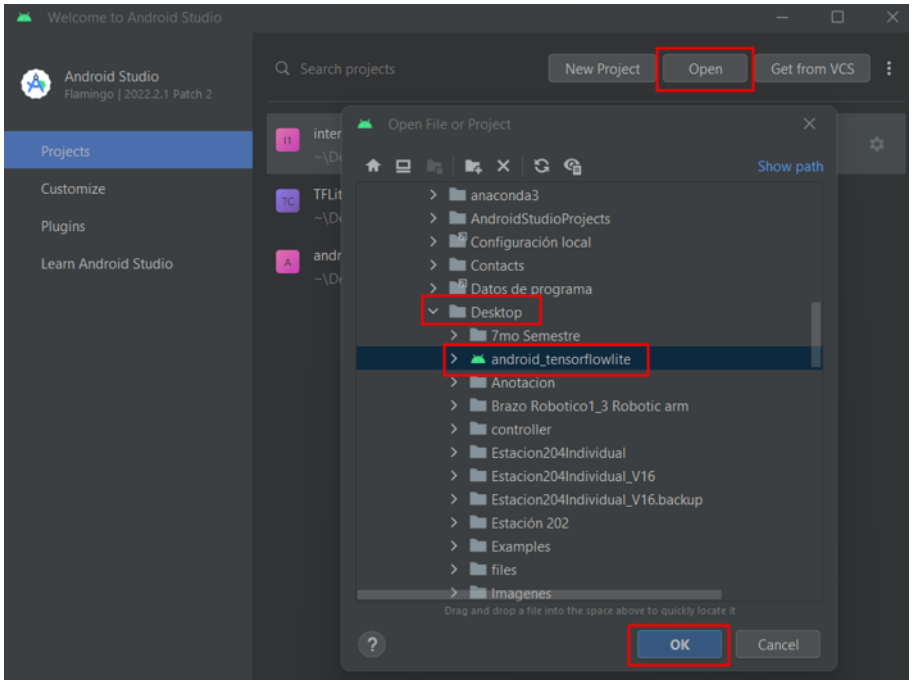

Figura 2.22: Pasos para abrir un proyecto existente en Android Studio.

5. Para cargar el modelo previamente entrenado en la actividad 3.2 con un formato *.tflite*, es necesario dirigirse a la opción "Project"situada a lado izquierdo de la pantalla. Esto

desplegará un panel con las carpetas del proyecto. Para cargar el modelo, ve a la carpeta *app* y dentro de ella, ubica la carpeta *assets*. Sobre ella hacemos click derecho y seleccionamos la alternativa *Open In* > *Explorer*. Esto abrirá la ubicación de la carpeta donde se debe a guardar el modelo entrenado. En este caso de estudio, el modelo se llama *model_fp16_meta.tflite*. La figura 2.23 muestra los pasos a seguir.

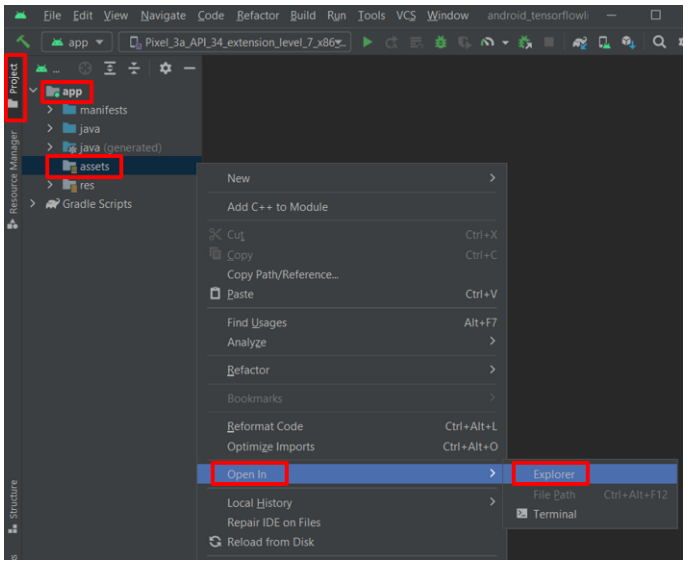

Figura 2.23: Actividades para cargar el modelo previamente entrenado.

6. Ahora, es necesario instanciar el archivo. Para lo cual, en el panel izquierdo, sitúa la carpeta *java*. Luego, selecciona la primera opción y abre el archivo llamado *ObjectDetectoHelper.kt*. En el documento, ubica la linea 85 y escribe el nombre del modelo cargado anteriormente. En la figura 2.24 se muestra los pasos descritos.

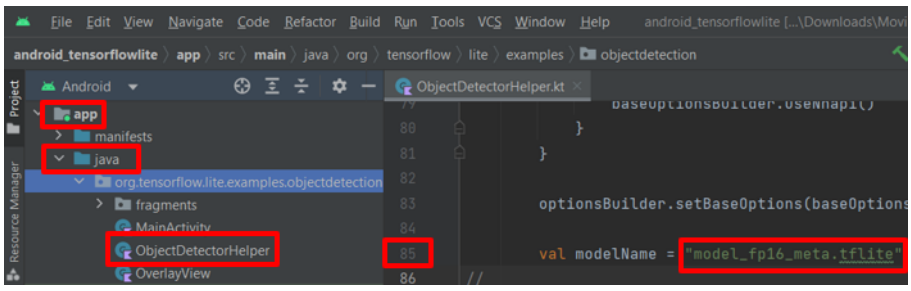

Figura 2.24: Instanciamos el modelo cargado en la carpeta *assets*.

Hasta este punto ya es factible verificar el funcionamiento de la aplicación, puesto que

es posible compilar el código y examinar el procesos de identificación por medio de un dispositivo móvil.

7. Para poder diseñar y programar la interfaz complementaria de la aplicación, es necesario de la ayuda de un diagrama de funcionamiento, esto lo pueden verificar en la figura 2.25. En donde se detalla las actividades y condiciones que debe cumplir la aplicación. Esta debe constar de dos interfaces gráficas. En la pantalla principal, el funcionamiento a seguir es que al iniciar la aplicación, las configuraciones, la cámara, y modelo de identificación se activan, es decir, el sistema esta esperando identificar una de las especies entrenadas. Si se detecta una de las especies, el botón denominado *Obtener propiedades* conduce a la interfaz secundaria. Dependiendo de la planta identificada los elementos de esta pantalla se actualizan automáticamente. Si no se ha identificado nada, entonces el usuario en la pantalla inicial debe desplegar el panel de configuraciones, reducir el umbral y el número de resultados, y luego restaurar el panel de configuraciones a su forma inicial, puesto que es un panel deslizante.

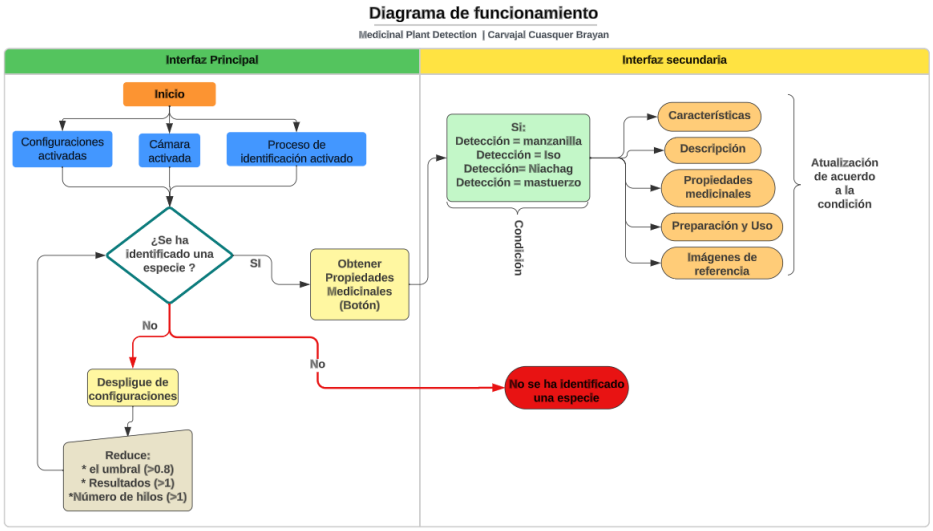

Figura 2.25: Diagrama de funcionamiento de aplicación móvil.

8. Como se pudo visualizar en la figura 2.25, la aplicación consta de dos interfaces. El diseño de la interfaz principal comienza solicitando el permiso del uso de la cámara al usuario. En caso de que el permiso sea garantizado, se continua con la actividad y si es denegado, la sesión simplemente finaliza. El código necesario para esto se muestra en la figura 2.26, donde se evidencia el uso del sistema de gestión de permisos proporcionado por el

entorno de desarrollo Android. Cabe destacar que, al autorizarse el uso de la cámara, se llama inmediatamente a la función *navigateToCamera*, guiándolos así a la activación de la cámara del dispositivo.

```
class PermissionsFragment : Fragment() {

    private val requestPermissionLauncher =
        registerForActivityResult(ActivityResultContracts.RequestPermission()
        ) { isGranted: Boolean ->
            if (isGranted) {
                Toast.makeText(context, text: "Permission request granted", Toast.LENGTH_LONG).show()
                navigateToCamera()
            } else {
                Toast.makeText(context, text: "Permission request denied", Toast.LENGTH_LONG).show()
            }
        }
```

Figura 2.26: Gestión de permisos para el uso de la cámara.

9. La interfaz principal no solo incluye el fragmento de la cámara, sino que también se vincula los código denominados *OverlayView.kt y ObjectDetectoHelper.kt*. En el primer archivo se encuentra toda la programación relacionada con la identificación de las plantas. Es decir, en este código se sitúan las variables donde se almacenan la información de las especies detectadas, así como también la construcción de las cajas delimitadoras que rodean al elemento. Y el segundo documento se detalla toda la programación que configura al botón deslizante situado en el inferior de la pantalla, el cual despliega al menú de configuraciones para gestionar el umbral, el número de resultados esperados y la red neuronal empleada sobre el algoritmo de identificación. La figura 2.27 muestra el diseño de la pantalla contenida en el archivo *fragment_camara*.

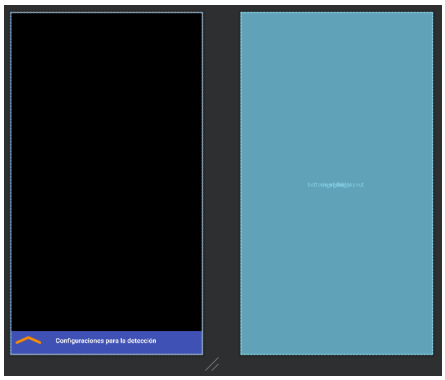

Figura 2.27: Diseño de la interfaz principal en el archivo *fragment_camara.xml*.

10. Además, se muestra información relevante sobre la especie identificada y el porcentaje de confianza sobre el 100 % que existe en la detección. Esto se lo realizó en la actividad principal del entorno, denominado *activity_main.xml*. Cabe recalcar que para presentar la especie identificada y el porcentaje de confianza se extrae las variables denominadas *labely y percentage* del archivo *Overlayview.kt*. En la figura 2.28 se muestra el diseño detallado en este ítem. Adicionalmente, este bloque contiene un botón denominado *Obtener propiedades* que dirige a la pantalla secundaria.

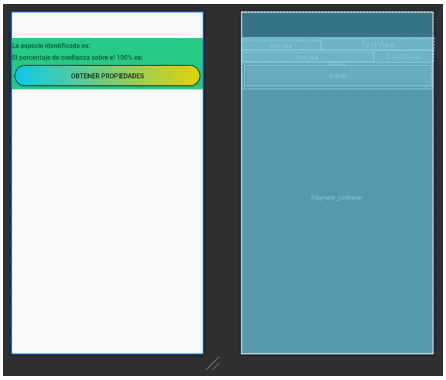

Figura 2.28: Información complementaria en el diseño en la interfaz principal por medio del archivo *activity_main.xml*.

11. Para configurar las características de la identificación, por medio del botón deslizante es posible acceder al panel de configuraciones, en el cual se puede aumentar o disminuir el umbral desde 0 % hasta 1 %, el número de resultados desde 1 hasta 5, la cantidad de hilos o procesos en ejecución desde 1 hasta 4; seleccionar la tarjeta gráfica y el modelo entrenado. Todos estos campos se encuentras programados en el archivo *info_bottom_sheet.xml*.

12. La pantalla secundaria depende en gran medida de los campos desarrollados en el ítem 10, puesto que la información presentada en está interfaz esta sujeta al cumplimiento de ciertas condiciones. Es decir, cuando la especie identificada sea manzanilla, actualiza la interfaz con el texto 1; sí la detección es Iso, muéstrame el texto 2 y así sucesivamente con las otras dos especies. El enlace para acceder a esta funcionalidad es el botón denominado *Obtener propiedades.*

    La lámina esta compuesta por cinco campos que son actualizados automáticamente por la condición descrita anteriormente, estos apartados están compuestos por los siguientes detalles.

Figura 2.29: Panel de configuraciones por medio del botón deslizante.

- **Campo 1:** *Características.*
  - Nombre científico.
  - Tipo de planta.
  - Altura.
  - Tamaño de la flor.
  - Hábitat.
- **Campo 2:** *Descripción.*
  - Nombres comunes.
  - Descripción.
- **Campo 3:** *Propiedades medicinales.*
  - Dolor (Articulaciones y garganta).
  - Infecciones.
  - Respiratorio.
  - Piel (inflamaciones o golpes).
  - Renal - Urológico.
  - Neurológico.
  - Anti-inflamatorio.
  - Fiebre.
  - Muscular.
- **Campo 5:** *Preparación y uso.*

- Parte empleada.
- Infusión o Tópico.

- **Campo 4:** *Imágenes de referencia.*

La figura 2.30, muestra la pantalla secundaria con las disposiciones mencionadas.

Figura 2.30: Interfaz secundaria con sus respectivos campos de información.

2.3.4. Etapa 4

- Validar el funcionamiento del algoritmo obtenido para la detección de plantas medicinales.

En la última etapa, se ejecuta la validación del sistema por medio de una matriz de confusiones, es decir, se verifica si el algoritmo creado identifica correctamente las plantas seleccionadas. Además, se determina si la interfaz de la aplicación es amigable y fácil de comprender para el usuario.

Actividad 4.1: **Validación del modelo entrenado mediante el uso de la matriz de confusiones.**

Con el propósito de verificar si el algoritmo lleva a cabo una identificación correcta, se emplea una matriz de confusión. Para esto, se utiliza el 20 % denominado como *validation* en la división realizada en la etapa 2, que se encuentra en el punto 2.3.2, específicamente en el ítem 7. Los pasos a seguir para cumplir con el último objetivo específico son los siguientes.

1. Dado que previamente se había separado el 20 % de la base de datos en el entorno donde se realizó el entrenamiento de la red neuronal. Utilizaremos este porcentaje equivalente a 820 imágenes, para validar el rendimiento del modelo. En primer lugar, se realizará la conversión de las etiquetas de cada imagen a un archivo en formato *.cvs*. La figura 2.31 ilustra el código empleado para este proceso.

Tranformamos a las etiquetas reales en un formato .cvs

```
import os
import glob
import csv
import xml.etree.ElementTree as ET

# Ruta de la carpeta que contiene los archivos XML de etiquetas reales
etiquetas_reales_folder = "/content/gdrive/MyDrive/modelo_entrenar/datos/ConfusionM

# Nombre del archivo CSV de salida para las etiquetas reales
csv_etiquetas_reales = "/content/gdrive/MyDrive/modelo_entrenar/datos/ConfusionMatr

# Obtiene la lista de rutas de archivos XML de etiquetas en la carpeta
etiquetas_reales_files = [f for f in glob.glob(os.path.join(etiquetas_reales_folder

# Abre el archivo CSV en modo escritura
with open(csv_etiquetas_reales, mode='w', newline='') as file:
    writer = csv.writer(file)

    # Escribe la cabecera del archivo CSV
    writer.writerow(["Imagen", "Etiquetas Reales"])

    # Procesa cada archivo XML de etiquetas
    for etiquetas_file in etiquetas_reales_files:
        # Parsea el archivo XML
        tree = ET.parse(etiquetas_file)
        root = tree.getroot()

        # Encuentra todas las etiquetas "object" que contienen etiquetas reales
        etiquetas = [obj.find("name").text for obj in root.findall("object")]
```

Figura 2.31: Recopilación de las etiquetas reales en un archivo denomina *Etiquetas reales*.

Además, es fundamental destacar que las etiquetas comprenden un total de 200 imágenes para las especies de: ñachag, iso y mastuerzo. Y finalmente la manzanilla está representada por 220 fotografías. El documento en formato *.cvs* muestra la información en el cual se detalla el nombre ó número de la imagen y la etiqueta real.

2. El siguiente paso consiste en obtener las etiquetas predichas. Para lo cual, es necesario cargar el total de las imágenes destinadas para validación y el modelo entrenado. Además, es importante señalar que se va a establecer un umbral 100 %, con el objetivo de que las identificaciones reflejen valores precisos. Los resultados de cada detección se registrarán en un nuevo archivo denominado *Etiquetas Predichas* y se almacenará en la carpeta situada en la nube. En la figura 2.32 , se puede evidenciar el proceso a seguir, y también

se adjunta el enlace hacia el repositorio de Github donde se encuentra el código descrita. Repositorio: *https://github.com/SpedyGo/Object-detection-with-TensorFlow*

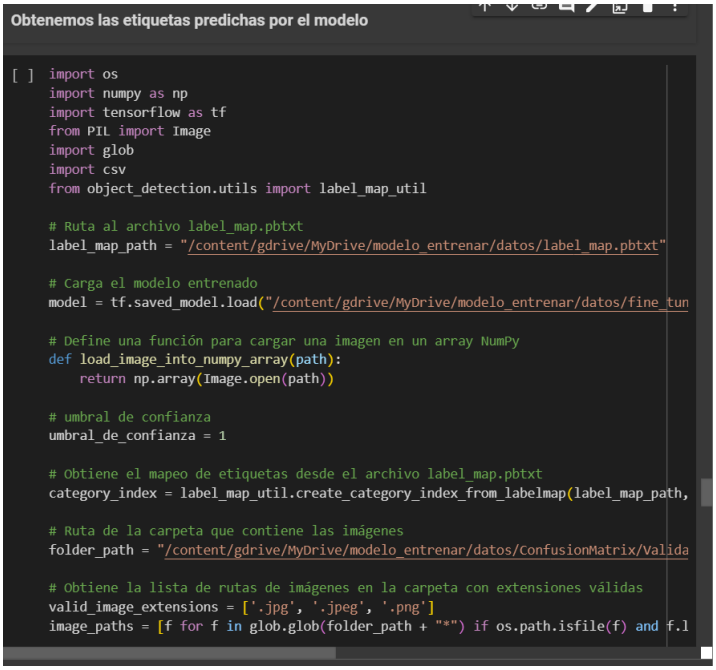

Figura 2.32: Obtención de las etiquetas predichas por el modelo entrenado con un porcentaje de umbral del 100 %.

3. Una vez obtenidas las etiquetas reales y las predichas, se procede al cálculo de la matriz de confusión, así como a la evaluación del rendimiento del modelo mediante diversas métricas. Es relevante destacar que la cantidad de información debe ser consistente en ambos archivos, ya que esto generará una matriz cuadrada. Para agilizar este proceso, es necesario importar dos librerías: la primera, *from sklearn.metrics import confusion_matrix*, es utilizada para la matriz, y la segunda, *from sklearn.metrics import classification_report*, se emplea para obtener los reportes de evaluación. En la figura 2.33 se muestra el desarrollo, donde se crea una lista de todas las etiquetas registradas, y mediante el uso de la librería, se realiza una comparación de todas las etiquetas, agrupándolas por especie.
   El objetivo de esto es que todas predicciones coincidan con las etiquetas reales, puesto que de esa forma se comprueba que el modelo funciona correctamente.

4. Para calcular las métricas de evaluación de manera global, es decir, donde son evaluadas todas las variables, es necesario el uso de la siguiente librería: *from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score*. Con esta biblioteca y con las

listas de etiquetas reales y predichas es posible validar el rendimiento del modelo. La figura 2.34 muestra el procesos desarrollado.

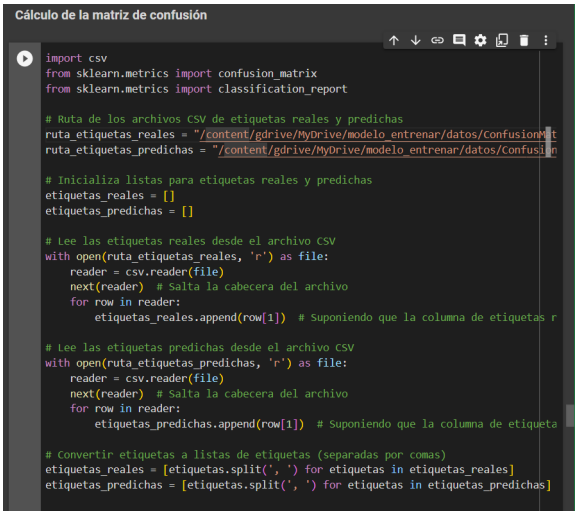

Figura 2.33: Cálculo de matriz de confusión.

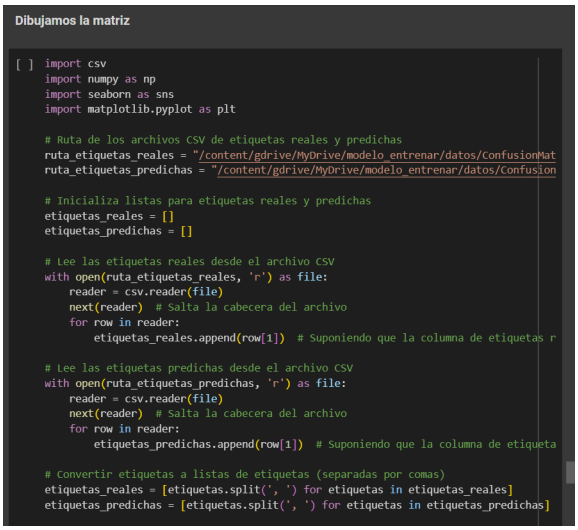

Figura 2.34: Evaluación global del rendimiento del modelo por medio de las métricas.

Capítulo III

Resultados y análisis.

En el contexto de este capítulo, se examinan los resultados alcanzados en el desarrollo de este trabajo de titulación. Estas respuestas satisfacen los objetivos específicos establecidos al comienzo de la investigación y, además abordan la problemática identificada en las personas que no llevan consigo un botiquín durante actividades de recreativas al aire libre, como caminatas, camping, trekking, entre otros. La información recopilada es de gran relevancia, tanto en la construcción y entrenamiento de la red neuronal como el desarrollo de la aplicación móvil, puesto que, todos las especificaciones requeridas son abordadas mediante las actividades llevadas a cabo en le marco metodológico, resultando en la creación de aplicación final con la capacidad de realizar inferencias a partir de un modelo entrenado que identifica plantas medicinales y brinda las propiedades de las mismas. A continuación, se ejecuta un análisis de los resultados obtenidos.

3.1. Especificaciones de la aplicación resultante.

El algoritmo de detección y el desarrollo de la aplicación están sujetas a las siguientes especificaciones.

- **Objeto a identificar:** El sistema esta entrenado con una base de datos que esta compuesta por fotografías que muestran solo la parte superior de la planta, es decir, las flores. Según los estudios realizados en los capítulos I y II, este modelo detecta las siguientes especies: Manzanilla, Ñachag, Iso y Mastuerzo. El modelo entrenado esta vinculado a la base de datos que contiene la información necesaria para la detección.
- **Algoritmo de entrenamiento:** El código de desarrollo con el cual se entreno el modelo para la detección está disponible para el público en general, lo que significa que es de li-

bre acceso y se encuentra en el repositorio del autor: *https://github.com/SpedyGo/Object-detection-with-TensorFlow/deployments/github-pages*

- **Localización de las plantas a identificar:** De acuerdo a la formación detallada en el marco metodológico sobre la localización de las plantas, estas pueden ser situadas en Pichincha, Imbabura y Carchi.
- **Precisión de la identificación:** La aplicación incluye un panel de configuraciones, que permite ajustar la precisión según las necesidad del usuario. Sin embargo, para evitar confusiones con otras plantas presentes en el medio ambiente, se recomienda factor de al menos el 80 %. De esta manera, se evita proporcionar información incorrecta sobre las propiedades medicinales de las plantas registradas en el sistema.
- **Interfaz de la aplicación:** Tanto la interfaz principal como la secundaria no contienen actividades ni textos difíciles de comprender. Es decir, están diseñadas de forma organizada con la finalidad de proporcionar información clara de acuerdo a la especie que se ha identificado.
- **Teléfono y cámara del dispositivo móvil :** De acuerdo a las pruebas realizadas, el celular debe ser me gama media o alta. La cámara del teléfono, en la medida de lo posible, debe ofrecer una imagen de resolución media, ya que el modelo requiere una buena entrada de datos para su posterior procesamiento.
- **Velocidad de identificación:** El modelo entrenado para la identificación de plantas es eficiente en inferencia, lo que implica que la detección la realiza de manera instantánea. Además, cabe recalcar que esta diseñada para operar en tiempo real.

3.2. A nivel de las propiedades y características de las plantas medicinales.

Imbabura es un provincia que, gracias a su clima cálido, templado y frió, alberga una amplia variedad de plantas medicinales con propiedades útiles para asistir cualquier percance en actividades al aire libre. Como resultado de esta investigación realizada y de la entrevista con una persona experta en el área de la medicina ancestral, se ha llegado a un consenso de que las especies como: la manzanilla, la ñachag, el iso y el mastuerzo poseen las propiedades detalladas en el capítulo I.

Además, tras revisar el " Estudio etnobotánico de plantas medicinales utilizadas en tres cantones de la provincia de Imbabura, Ecuador"[22], documento del cual se obtuvo información relevante para el primer capítulo, y con base en la entrevista, es posible aportar al estudio las siguientes plantas:

- *Trébol Blanco (Analgésico):* Propiedades medicinales que calman los dolores causados por el cáncer.
- *El chuzo (Analgésico):* Calma los dolores de muela.

3.3. A nivel del algoritmo de entrenamiento.

El resultado final para el segundo objetivo específico es la obtención del modelo entrenado con una arquitectura SSD MobileNet V2 con el cual se logra identificar las cuatro plantas medicinales. En todo este proceso de entrenamiento se obtuvieron resultados muy importantes que caracterizan el rendimiento del algoritmo y a su vez son analizados a continuación.

3.3.1. Entrenamiento de la red.

El entrenamiento de la red neuronal se llevó a cabo con un tamaño de lote (batch size) de 35 imágenes y un total de 60,000 pasos. Esto significa que en cada paso, el algoritmo recibía 35 fotografías para la extracción de las características en cada capa de convolución, con el objetivo de obtener los rasgos más detallados de las especies a entrenar. Con estos valores, se completaron un total de 1714 épocas, lo que implica que se procesaron todas las imágenes en la base de datos 1714 veces. Por otro lado, en la consola mientras la red neuronal aprendía, se observó que la tasa de aprendizaje disminuyó de manera gradual. Es importante destacar que a partir del paso 50,000 se evidenció que el ritmo de aprendizaje fue igual a 0. Esto se puede verificar el figura 3.1.

Figura 3.1: Ritmo de aprendizaje igual a 0. La red ya no recibía datos nuevos.

Para complementar la información sobre la tasa de aprendizaje, mediante la aplicación de la herramienta de Tensorboard se obtiene diferentes imágenes escalares sobre el procesos de ejecución en el entorno de programación. En la figura 3.2, se observa que el modelo comienza con una pérdida inicial de 0.03. Luego, tras 3000 pasos, la pérdida aumenta a 0.08, indicando que el modelo buscó rápidamente una solución óptima para acercarse a las predicciones reales. En otras palabras, el sistema optó por una solución rápida para abordar el problema. Una vez que entrenamiento está bajo control o las predicciones se acercan a las reales, las pérdidas de la taza de aprendizaje comienzan a disminuir, y como consecuencia, el modelo ejecuta más épocas de entrenamiento.

Este proceso tuvo una duración de 7 horas y se llevó a cabo utilizando una tarjeta gráfica GPU T4 proporcionada por Google Colab.

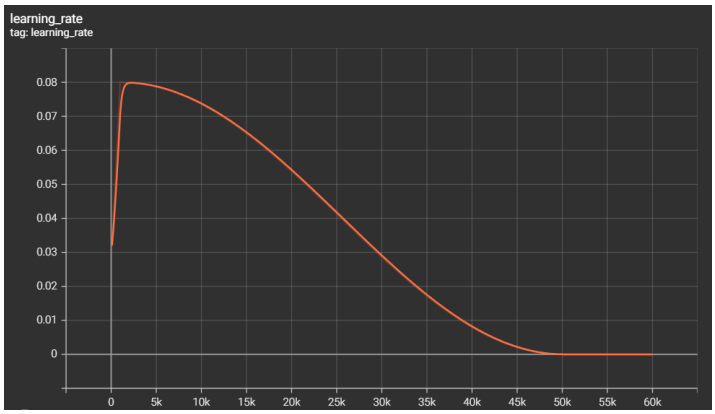

Figura 3.2: Ritmo de aprendizaje decreciente.

3.3.2. Perdidas del modelo.

Gracias a la biblioteca que permite evaluar al modelo y con base en los resultados del entrenamiento, es posible obtener cuatro gráficas escalares que representan las diferentes pérdidas que obtuvo el modelo. A continuación, se analiza cada una de las curvas exponenciales en referencia a la figura 3.3.

- **Perdidas por clasificación** *(classification_loss)*: Este tipo de pérdida mide la diferencia que existe entre las predicciones realizadas por el modelo en relación a las etiquetas reales. Como se puede verificar en la imagen, las pérdidas son altas al inicio del entrenamiento; conforme transcurren los pasos, estas disminuyen al punto de que a los 50,000

pasos el valor era de 0,037 y al culminar el entrenamiento, el valor concluyó en 0,036. Lo que significa que las predicciones que las predicciones realizadas por modelo no presentan mayor discrepancia con las etiquetas reales.

- **Perdidas por localización** *(localization_loss)*: Mide la divergencia que existe entre la creación de las cajas delimitadoras del objeto a identificar en cada imagen por el modelo entrenado con los cuadros delimitados en cada planta por el usuario a través del software *LabelImg*. En otras palabras, verifica la discrepancia que existe entre las coordenadas de las etiquetas reales y las predichas. Como resultado, se obtiene una diferencia de 0,01, lo cual indica que las coordenadas tienden a variar un rango de ± 0,01.

- **Perdidas por regularización** *(regularization_loss)*: La representación gráfica indica la presencia o ausencia de sobre entrenamiento en los datos. En caso de que se identifique, se aplica una penalización para evitar que el aprendizaje de la red sea excesivamente complejo. De acuerdo con la figura, es válido afirmar que al comienzo del entrenamiento había sobre ajuste. Gracias a las penalizaciones, esto se reduce y, en última instancia, se alcanza un valor de 0.05, indicando que el modelo no contiene información perjudicial para la identificación de las variables. Cabe resaltar que cuando un modelo experimenta sobre entrenamiento, el sistema puede experimentar una desaceleración en sus predicciones, lo cual es perjudicial, ya que puede saturar el sistema.

- **Perdidas totales** *(total_loss)*: Es el valor más relevante, ya que se determina mediante una suma ponderada entre las pérdidas por clasificación y localización, y además es un indicador que demuestra si el modelo entrenado tiene precisión o no. Para ello, en la última gráfica de la figura 3.3, se puede evidenciar una pérdida de 0.4.

En definitiva y según la red neuronal utilizada (SSDMOBILENET V2), el valor de 0.5 como perdida total indica una alta inferencia y buena precisión en el modelo. Es importante destacar que todos los procesos de pérdidas comienzan con valores altos y, con el transcurso de los pasos realizados, estos disminuyen y se sitúan por debajo de 1, lo que significa que el modelo es aceptable, ya que el rango de un entrenamiento va de 0 a 1. Esta es una de las primeras validaciones del modelo, la cual demuestra que el funcionamiento de identificación se lo realiza con exactitud. Posteriormente se valida el proceso de detección mendicante la evolución del entrenamiento y la creación de una matriz de confusión.

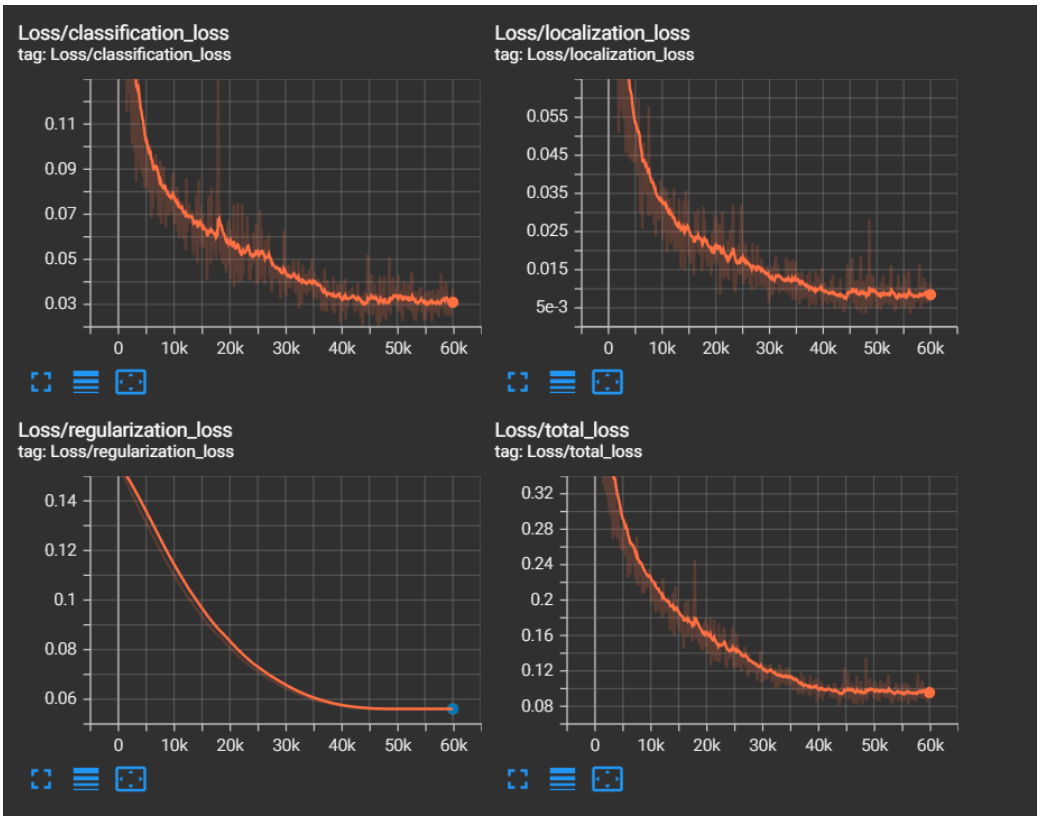

Figura 3.3: Perdidas del sistema de visión artificial

3.3.3. Evaluación del modelo entrenado.

Para la evaluación del modelo a través de la codificación, se llevó a cabo de dos maneras. La primera fue mediante el uso de TensorBoard, que permite evaluar el progreso del entrenamiento a lo largo de los pasos. El modelo proporcionaba continuamente resultados sobre cómo la detección mejoraba; es decir, cuando la red neuronal había completado 5000 pasos, la identificación de las especies no era precisa, dado que pocas es las flores que tenían 100 % de precisión, pero al finalizar el proceso completo (60000 pasos), los resultados cambiaron significativamente. Esto se puede verificar en la figura 3.6

En la parte izquierda de la figura 3.6, se observa que en cada flor, los resultados de la detección se encuentran en un rango que va del 50 al 80 %. Una vez que el modelo ha completado todos los procesos de entrenamiento, los resultados se presentan al 100 %, indicando que la red está entrenada para la identificación de las especies mencionadas seleccionadas.

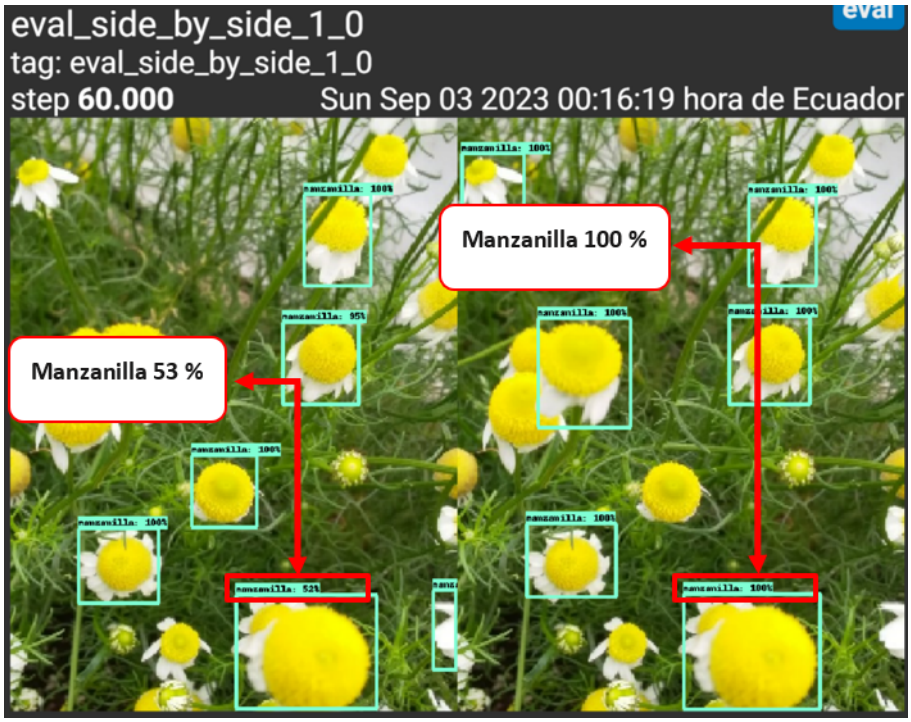

Figura 3.4: Identificación de la manzanilla a los 5000 y 60000 pasos *(steps)*

La segunda manera de validar el rendimiento de la red neuronal es mediante la verificación de la inferencia del modelo. Para ello, se carga una imagen cualquiera, se establece un umbral del 100 %, es decir, el porcentaje de confianza, y se solicita, mediante código, que el modelo detecte la especie de la fotografía cargada. Como se puede observar en la figura 3.5, el porcentaje mostrado es del 100 %, y, en realidad, la especie mostrada contiene la etiqueta identificada; en este caso corresponde a la etiqueta real denominada, Niachag.

Este resultado contribuye para el segundo objetivo específico, el cual consiste en el entrenamiento de la red neuronal. Como se ha comprobado en los dos casos anteriores, el modelo opera correctamente. Las pérdidas que presenta están dentro de los rangos establecidos para los procesos de entrenamiento, indicando que la solución propuesta aborda satisfactoriamente el objetivo establecido.

Es importante destacar que tanto las actividades como la obtención de los resultados están vinculadas al modelo en cascada. Esto posibilita la organización de los procesos propuestos y, sobre todo, impide avanzar hasta que los resultados de cada actividad cumplan con las especificaciones u objetivos establecidos.

Figura 3.5: Inferencia del modelo entrenado con un resultado del 100 % en la identificación de la especie.

3.4. Modelo en formato *.tflite*

El resultado más destacado de este apartado es la obtención del modelo optimizado en formato *.tflite*. Este archivo facilita la identificación de las plantas medicinales a través del teléfono inteligente en tiempo real. En otras palabras, se utiliza para llevar a cabo la inferencia y detectar las cuatro especies, ya que contiene la información esencial, como la arquitectura, los metadatos, el formato de entrada y salida, las optimizaciones específicas para el formato *.tflite*, y las operaciones de inferencia necesarias para las predicciones. En la figura 3.6 se presenta la carpeta llamada *tflite*, la cual está compuesta por el modelo y dos archivos adicionales en formato de punto flotante, uno en 32 y otro en 16 bits. Se trabaja específicamente con este último debido a que es liviano y no afecta el rendimiento del sistema operativo del celular.

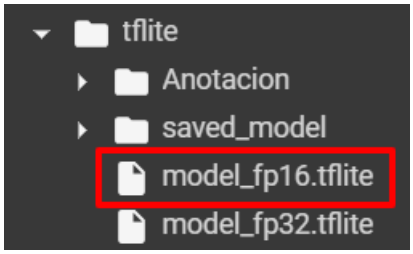

Figura 3.6: Modelo para la detección denominado *model_fp16.tflite*

3.5. Implementación del sistema de visión artificial como aplicación móvil

Como resultado del tercer objetivo específico, se obtiene la creación de la aplicación móvil mediante Android Studio, con la implementación del modelo obtenido para la identificación de las especies en formato *.tflite*. Una vez que se detecte cualquiera de las cuatro plantas entrenadas en la red neuronal, el sistema muestra las características, la descripción, las propiedades medicinales, el modo de uso y, finalmente, imágenes de referencia.

3.5.1. A nivel de la pantalla de inicio.

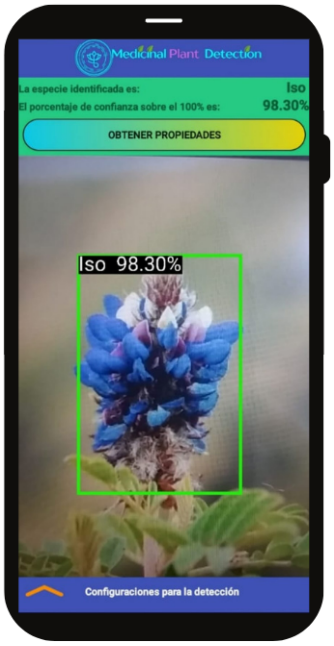

Figura 3.7: Pantalla de inicio de la aplicación móvil *Medicinal Plant Detection*.

Como se puede apreciar en la figura 3.7, la pantalla de inicio muestra el resultado principal del objetivo específico en ejecución. Además se evidencia la implementación del modelo entrenado con la identificación en tiempo real, es decir, con un tiempo de inferencia bastante bajo para la detección de una de las especies con la denominación *Iso*. También se muestra el porcentaje de confianza en el reconocimiento , ya que estos valores reflejan la similitud entre la especie identificada y los metadatos registrados en el modelo de identificación. Cabe destacar que el porcentaje de confianza debe situarse en el intervalo del 80 al 100 %. Es importante

señalar que estos valores, conocidos como umbral o en inglés como Threshold, son ajustables desde el panel de configuraciones para la detección, variando desde 0 hasta el 100 %. No obstante, se recomienda realizar los ajustes dentro del rango mencionado anteriormente para evitar confusiones con otras especies y obtener mejores resultados.

Dado que el modelo es ejecutable en tiempo real y tanto la etiqueta como el porcentaje varían indeterminadamente.Por esta razón se proporciona un cuadro de texto en la parte superior que muestra la información resultante del modelo entrenado. Este registro de información permite actualizar la presentación en la pantalla de detalles.

El panel deslizante para las configuraciones permiten ajustar los resultados presentados, este esta compuesto por los siguientes campos.

- ***Tiempo de inferencia:*** Muestra el tiempo transcurrido hasta la identificación de una especie en tiempo real; este campo se mide en milisegundos. Dado que la aplicación está diseñada para abordar situaciones que requieran el uso de las propiedades medicinales de cada planta y por esta razón el tiempo es fundamental, ya que contribuye de forma breve se puede ayudar a reducir dolores y molestias en las personas afectadas.

- ***Umbral:*** Es el porcentaje de confianza que se desea tener en cada identificación. Como se mencionó anteriormente, estos valores deben estar dentro del rango del 80 al 100 %, ya que la información de detección en este intervalo denota mayor precisión.

- ***Resultados:*** Es la cantidad de resultados que se desea visualizar en la pantalla. Esta configuración está programada para mostrar desde 1 hasta 5 datos; es decir, dependiendo de la disposición, el sistema mostrará la información requerida.

- ***Número de hilos:*** Es la cantidad de procesos que realiza el dispositivo para la identificación de alguna especie. Esto realmente depende de las características del dispositivo con el que se esté trabajando. Se recomienda configurarla en 2 o 3, especialmente si es un teléfono de gama baja o media, ya que la detección se compila de la mejor manera y además respeta las configuraciones visualizadas anteriormente. Si el celular es de gama alta, entonces la configuración del número de hilos debería ser el valor máximo.

- ***Tarjeta:*** Permite elegir la tarjeta gráfica para llevar a cabo la inferencia. Hay dos tipos: la primera es la unidad de procesamiento central, también conocida por sus siglas como CPU, y la segunda es la unidad de procesamiento gráfico, conocida como GPU. Esta elección depende del rendimiento del dispositivo móvil. Claro está que la aplicación de la GPU proporciona mejores resultados en comparación con la CPU, pero esto no afecta la precisión de la información mostrada.

- ***Modelo:***Cumple la función de seleccionar qué red neuronal se utilizará para la identificación. Como se ha indicado en el capítulo de la metodología, actualmente se está utilizando exclusivamente la red neuronal SSDMOBILENET V2 que es con la cual se realizo el entrenamiento del modelo para la detección.

En la figura 3.8, se muestran todos los campos mencionados correspondientes al panel de configuración. Es importante destacar que las configuraciones son únicas e independientes para cada usuario de la aplicación.

Figura 3.8: Panel de configuraciones para la identificación de plantas.

3.5.2. A nivel de la pantalla de detalles.

La pantalla secundaria se compone de los campos detallados en el capítulo II, específicamente en el punto 2.3.3, subactividad 12. Estos atributos se actualizan automáticamente según la especie identificada. Para acceder a esta pantalla, se utiliza el botón denominado *Obtener propiedades*. Es importante destacar que toda la información presentada en la pantalla guarda una estrecha relación con la revisión bibliográfica del capítulo I, así como con la información obtenida mediante las entrevistas realizadas.

La interfaz adicional inicialmente presenta campos vacíos; una vez que el modelo realiza la inferencia con una de las especies entrenadas, todos los ítem se actualizan con los datos

correspondientes. Es relevante destacar que, al ser una aplicación destinada a ayudar a personas que no llevan consigo un botiquín como medida de emergencia, es necesario que la información proporcionada sea concisa. Por lo tanto, las propiedades medicinales y su uso se detallan de manera clara. Además, en la parte final se brinda imágenes de referencia con la cual se puede verificar si el modelo realizo la identificación correctamente.

A continuación en la figura 3.9, se muestran los campos con información detallada de acuerdo a la especie que se ha identificado en la figura 3.8, en este caso *mastuerzo*.

Figura 3.9: Pantalla secundaria de la aplicación móvil *Medicinal Plant Detection*.

3.6. Validación del funcionamiento del algoritmo de visión artificial.

Para validar el funcionamiento del algoritmo utilizado en la identificación de plantas medicinales, se emplea la herramienta conocida como *matriz de confusión*. Esto se realiza con el propósito de organizar los resultados de la clasificación en un gráfico de calor, permitiendo evaluar todas las variables de manera independiente frente a las otras. Es decir, se calculan los verdaderos positivos, que se encuentran en la diagonal principal, y los verdaderos negativos, en las celdas restantes de la matriz. De esta manera, se busca verificar si el modelo presenta alguna confusión entre las demás plantas seleccionadas. Cabe destacar que, para que el sistema

esté validado correctamente, la mayoría de las fotografías destinadas para la validación deben situarse en la diagonal principal y solo ahí es correcto afirmar que el sistema esta identificando correctamente.

3.6.1. Matriz de confusión.

Para validar el funcionamiento del modelo entrenado, es esencial emplear el 20 % del total de la base de datos, lo que equivale a 820 imágenes. Esta cantidad se distribuye de la siguiente manera: 220 para la categoría de manzanilla y las 600 imágenes restantes se dividen equitativamente en 200 para cada una de las especies, a saber: ñachag, iso y mastuerzo.

El resultado clave para el cuarto objetivo específico es la matriz de confusión presentada en la figura 3.10. Cada columna representa los valores predichos por el modelo con un umbral de confianza del 100 %, mientras que cada fila representa los valores reales, los cuales han sido etiquetados manualmente mediante el software *LabelImg*.

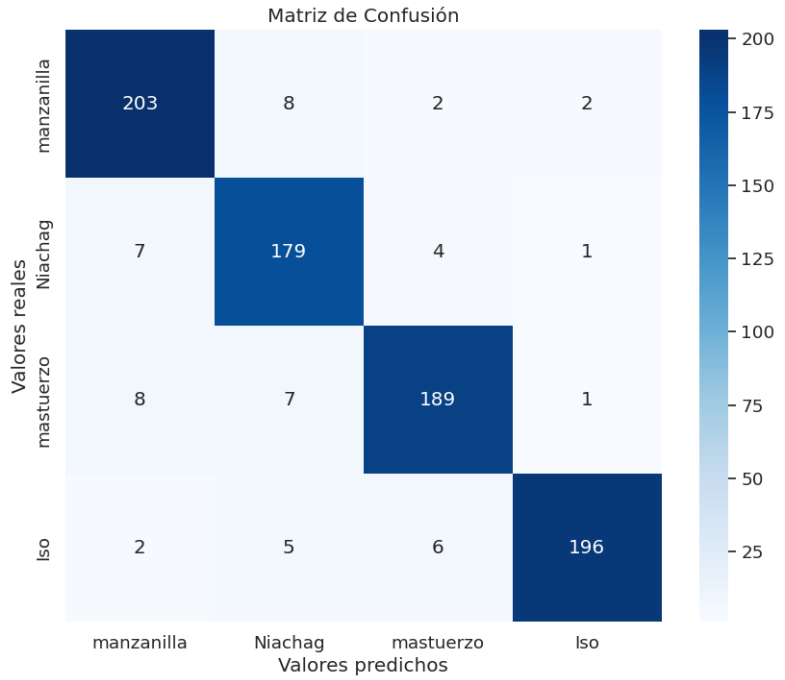

Figura 3.10: Validación del modelo entrenado por medio de la Matriz de confusiones.

De acuerdo al gráfico de calor de la figura 3.10, es posible realizar el siguiente análisis.

- ***Diagonal principal:*** La diagonal principal alude a las predicciones acertadas del modelo. Como se destacó previamente, la categoría de manzanilla constaba de 220 imágenes,

mientras que las otras categorías comprendían 200 cada una. En este contexto, se puede derivar la siguiente tabla informativa, donde se especifica la cantidad de imágenes predichas correctamente y aquellas que el modelo interpretó de manera equivocada.

Tabla 3.1: Análisis de los resultados de la diagonal principal.

Variable	**# de Validación**	**# Correctas**	**# Erróneas**	
Manzanilla	220	203	17	Imágenes
Ñachag	200	179	21	Imágenes
Mastuerzo	200	189	11	Imágenes
Iso	200	196	4	Imágenes
Total	820	767	53	Imágenes

Como se aprecia en la tabla 3.1, la diagonal principal de la matriz está integrada por 767 fotografías que fueron predichas correctamente. Esto implica que la gran mayoría de las imágenes en la base de datos fueron clasificadas de manera acertada por el modelo entrenado. Es relevante señalar que, considerando 820 datos como el 100 %, las 767 figuras corresponden al 93.5 % resultado que hace referencia a la precisión del sistema . Este porcentaje refleja una alta capacidad del modelo para la detección de especies.

El porcentaje de datos detectados de manera incorrecta es del 6.5 %. Es relevante destacar que estos resultados se encuentran fuera de la diagonal principal y pueden estar constituidos por falsos positivos o falsos negativos. Esta observación conlleva a la siguiente deliberación:

- Dado que el modelo realizó predicciones incorrectas, afirmando que una fotografía pertenecía a una clase específica cuando, en realidad, no lo era, o no logró identificar alguna planta cuando sí existía, surge la cuestión de qué factores influyeron en el modelo para que estas imágenes no se verificaran. Al analizar cada una de las imágenes, se determina que los factores que influyen en obtener resultados fuera de la diagonal principal están relacionados con la calidad de la imagen, la diversidad de fondos y colores, la superposición de objetos y, finalmente, el umbral que configura la precisión del algoritmo de identificación. Estos elementos provocaron cierta confusión en el modelo, aunque el porcentaje es significativamente bajo en comparación con las imágenes que sí logró identificar. Estas son causas que se pueden tomar en cuenta para mejorar le rendimiento de la red neuronal.

3.6.2. Métricas de evaluación para cada clase.

Para llevar a cabo la evaluación de cada clase de manera independiente, es crucial comprender todos los elementos de una matriz de confusión. Esta matriz está compuesta por términos específicos que son detallados a continuación [58]:

1. *Verdaderos positivos TP:* Muestra de imágenes que pertenecen a a la clase positiva que son identificadas de forma correcta.
2. *Verdaderos negativos TN:* Muestra de imágenes que pertenece a la clase negativa que se identifica correctamente.
3. *Falsos positivos FP:* Muestra de imágenes que pertenecen a la clase negativa pero que son identificadas erróneamente como correctas.
4. *Falsos negativos FN:* Muestra de imágenes que pertenecen a la clase positiva pero que son identificadas como negativas.

Además es necesario tener en cuenta los términos que van a ser evaluados, estos se describen a continuación [58]

- ***Exactitud (Accuracy):*** Representa la cantidad de imágenes que fueron clasificadas correctamente. La fórmula es la siguiente.

$$Exactitud = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.1}$$

- ***Precisión:*** Representa el número de muestras que se identificaron correctamente, es decir a la clase positiva. Su fórmula es la siguiente.

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

- ***Sensibilidad (Recall):*** Hace referencia al porcentaje de imágenes que se clasificaron correctamente. La fórmula es la siguiente.

$$Sensibilidad = \frac{TP}{TP + FN} \tag{3.3}$$

- ***F1-Score:*** Es una medida armónica entre la precisión y la sensibilidad, es decir, el equilibrio que tiene el modelo. La fórmula es la siguiente:

$$F1 - Score = \frac{2 * Precision * Sensibilidad}{Precision + Sensibilidad} \tag{3.4}$$

- ***Especificidad:*** Es el número de datos que fueron clasificados correctamente como verdaderos negativos. La fórmula para calcular esto es la siguiente.

$$Especifidad = \frac{TN}{FP + TN} \tag{3.5}$$

Utilizando la matriz de confusión global, se pueden derivar las métricas de evaluación específicas para cada clase. Para lograrlo, es esencial identificar los verdaderos positivos (TP) y negativos (TN), así como también los falsos positivos (FP) y negativos (FN), los cuales están detallados en la siguiente tabla.

Tabla 3.2: Estados de evaluación para cada clase.

Clase	**Verdaderos positivos**	**Verdaderos negativos**	**Falsos positivos**	**Falsos negativos**
Manzanilla	203	588	12	17
Ñachag	179	609	12	20
Mastuerzo	189	603	16	12
Iso	196	607	13	4

Después de identificar los datos individuales que conforman una matriz de confusión, es factible calcular las métricas de evaluación utilizando las fórmulas previamente detalladas. Los resultados obtenidos se presentan en la tabla siguiente.

Tabla 3.3: Evaluación del modelo entrenado por medio de las métricas.

Clase	**Exactitud**	**Precisión**	**Sensibilidad**	**F1- Score**	**Especificidad**
Manzanilla	0,964	0,944	0,922	0,961	0,98
Ñachag	0,960	0,937	0,899	0.92	0,981
Mastuerzo	0,965	0,921	0,940	0,932	0,974
Iso	0,979	0,937	0,980	0,94	0,979

Basándonos en los cálculos registrados en la tabla 3.3, donde se examinan las métricas para cada variable, se puede efectuar el siguiente análisis con respecto a los criterios de evaluación mencionados en relación con las clases pertinentes.

- En relación con el ítem de exactitud, el cual presenta la proporción de imágenes que fueron clasificadas correctamente como positivas o negativas, se puede afirmar que cada una de las especies fue categorizada con un valor superior al 0,96. Esto sugiere que el modelo es capaz de identificar y clasificar las plantas según las etiquetas registradas. En términos más sencillos, el 96 % de las predicciones son precisas.

- Respecto a la métrica de precisión, que evalúa la proporción de elementos clasificados como verdaderos positivos y falsos positivos, es válido afirmar que la manzanilla exhibe un valor de 0.94, la ñachag y el iso de 0.93, y finalmente el mastuerzo de 0.92. En términos generales, este criterio de valoración supera el 90 %, lo cual sugiere que hay una alta probabilidad de clasificación correcta para las plantas medicinales.

- En cuanto a los resultados obtenidos sobre la sensibilidad del modelo, que determina cuántos elementos positivos fueron identificados correctamente, se puede afirmar que la manzanilla, el mastuerzo y el iso presentan un porcentaje superior al 90 %, indicando así una eficiencia destacada en la identificación. Por otro lado, en la variable ñachag, en comparación con las demás, se observa un ligero descenso con un porcentaje del 89 %. Aunque esta diferencia es mínima, es esencial tener en cuenta todos los resultados obtenidos con la finalidad de evaluar de manera integral el rendimiento del modelo.

- Para evaluar el equilibrio entre precisión y sensibilidad, se utiliza la métrica denominada F1-score. Como se puede observar en la tabla 3.3, los porcentajes de las variables están por encima del 92 %, indicando que el modelo está equilibrado. Este equilibrio es crucial, ya que a medida que el porcentaje se acerca más a 1, el modelo demuestra un rendimiento superior y si el valor esta más alejado de 1 notoriamente esta sujeto a enfrentar complicaciones a la hora de identificar cualquier especie.

- La especificidad es un término que evalúa el rendimiento de la red neuronal para identificar elementos negativos. Para el caso de estudio todas las variables esta sobre el 97 %, lo que significa que el modelo tiene la capacidad para identificar si en las imágenes se encuentran o no los elementos entrenados.

Capítulo IV

Conclusiones, Recomendaciones y Trabajo a futuro

4.1. Conclusiones

- Imbabura desempeña un papel crucial en la medicina tradicional, ya que, según su ubicación, alberga plantas con propiedades medicinales muy beneficiosas. Estas plantas son valiosas para aquellas personas que participan en actividades recreativas. A partir de esta riqueza botánica es posible concluir que existen especies con características peculiares como su forma y los colores llamativos para la vista del ser humano, gracias a esto se ha creado una base de datos completa para el entrenamiento de un algoritmo.

- En relación con el entrenamiento de la red neuronal MobileNet V2 y el uso de la herramienta *tensorboard*, se puede concluir que el modelo exhibe pérdidas muy bajas, abarcando las pérdidas por clasificación, localización y regularización, todas ellas situadas por debajo de 0.05. Este resultado sugiere que el modelo cuenta con una alta capacidad de inferencia y precisión en la tarea de identificación.

- En lo que respecta al funcionamiento de la aplicación móvil en entornos naturales con la inferencia del modelo entrenado en formato *.tflite*, se puede concluir que cumple con el alcance planteado, puesto que la aplicación ejecuta la identificación de la plantas medicinales previamente entrenadas y proporciona instantáneamente información detallada, incluyendo características, descripción, propiedades, preparación y uso, junto con imágenes de referencia. Cabe destacar que esta información resulta de gran utilidad para abordar situaciones inesperadas en la naturaleza.

- La red neuronal demuestra un alto rendimiento para reconocer cuatro plantas medicinales: manzanilla, ñachag, iso y mastuerzo. Además, el algoritmo esta validado mediante una matriz de confusión, de la cual se obtiene que la confianza en la identificación en tiempo real es de un 93.5 %, subrayando la eficacia del modelo para proporcionar resultados precisos. Además, se destaca un tiempo de inferencia relativamente bajo, medido en milisegundos *(ms)*, lo que resalta la eficiencia del modelo en procesar y clasificar de manera rápida las plantas mencionadas. En relación con las métricas de evaluación, se puede concluir que la exactitud, precisión, sensibilidad, F1-score y especificidad superan el 90 %. Este resultado indica que el modelo cumple con los criterios de evaluación, demostrando una alta confiabilidad en cada identificación realizada.

4.2. Recomendaciones

- Se recomienda verificar que el porcentaje de confianza en la aplicación móvil dentro del panel de configuraciones esté por encima del 80 % antes de utilizarla. Esta medida tiene como objetivo asegurar una alta precisión en las identificaciones realizadas por el algoritmo.
- Cuando se haya identificado una planta medicinal, se recomienda realizar el proceso dos veces por motivos de seguridad. Esto garantiza que el algoritmo identifique y realice comparaciones con las imágenes de prueba cargadas en el sistema.
- Después de identificar una especie y acceder a la pantalla de información, se recomienda verificar que las características concuerden con la planta detectada. De esta manera, a pesar de la precisión del modelo, se evitan posibles confusiones que puedan surgir.
- La buena iluminación en el entorno donde se desea identificar las especies es crucial. Esto se lo hace con la finalidad de evitar confusiones con la identificación que realice el modelo.

4.3. Trabajo a futuro

Analizando los resultados obtenidos por medio del algoritmo de visión artificial que permite realizar la identificación de las especies por medio de la parte superior de su taxonomía, es decir las flores ; es correcto brindar los siguientes trabajos a futuro:

- Algoritmo de identificación de plantas medicinales empleando una red neuronal diferente a la que se esta usando en este trabajo de titulación.
- Algoritmo de visión de artificial enfocado a la identificación de hojas medicinales.
- Aplicación móvil enfocada al reconocimiento de plantas medicinales con la aplicación de un entrenamiento no supervisado.

Bibliografía

[1] S. Belliure, Barberá Víctor, J. Martínez, S. García, and A. Azorín, "El Equipo que necesitas para disfrutar del Senderismo - AlmaOutdoor."

[2] P. Brooks, "How Many Hikers Die Each Year? | Hikers University," 9 2022.

[3] L. C. Ngugi, M. Abdelwahab, and M. Abo-Zahhad, "A new approach to learning and recognizing leaf diseases from individual lesions using convolutional neural networks," *Information Processing in Agriculture*, vol. 10, pp. 11–27, 3 2023.

[4] R. Azadnia and K. Kheiralipour, "Recognition of leaves of different medicinal plant species using a robust image processing algorithm and artificial neural networks classifier," *Journal of Applied Research on Medicinal and Aromatic Plants*, vol. 25, p. 100327, 12 2021.

[5] Z. Wu, C. Liu, C. Huang, J. Wen, and Y. Xu, "DEEP OBJECT DETECTION WITH EXAMPLE ATTRIBUTE BASED PREDICTION MODULATION," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2022-May, pp. 2020–2024, 2022.

[6] A. Cardellicchio, F. Solimani, G. Dimauro, A. Petrozza, S. Summerer, F. Cellini, and V. Renò, "Detection of tomato plant phenotyping traits using YOLOv5-based single stage detectors," *Computers and Electronics in Agriculture*, vol. 207, p. 107757, 4 2023.

[7] H. F. Ng, C. Y. Lin, J. H. Chuah, H. K. Tan, and K. H. Leung, "Plant Disease Detection Mobile Application Development using Deep Learning," *Proceedings - International Conference on Computer and Information Sciences: Sustaining Tomorrow with Digital Innovation, ICCOINS 2021*, pp. 34–38, 7 2021.

[8] M. F. Mohamedon, F. Abd Rahman, S. Y. Mohamad, and O. Omran Khalifa, "Banana Ripeness Classification Using Computer Vision-based Mobile Application," *Proceedings of*

the 8th International Conference on Computer and Communication Engineering, ICCCE 2021, pp. 335–338, 6 2021.

[9] X. H. Yang, Z. J. Xi, J. P. Li, X. L. Feng, X. H. Zhu, S. Y. Guo, and C. P. Song, "Deep Transfer Learning-Based Multi-Object Detection for Plant Stomata Phenotypic Traits Intelligent Recognition," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 20, pp. 321–329, 1 2023.

[10] H. G. Jones, "What plant is that? Tests of automated image recognition apps for plant identification on plants from the British flora," *AoB PLANTS*, vol. 12, 12 2020.

[11] S. Anubha Pearline and V. Sathiesh Kumar, "Performance analysis of real-time plant species recognition using bilateral network combined with machine learning classifier," *Ecological Informatics*, vol. 67, p. 101492, 3 2022.

[12] C. Yang and H. Wei, "Plant Species Recognition Using Triangle-Distance Representation," *IEEE Access*, vol. 7, pp. 178108–178120, 2019.

[13] M. DAS, *CHAMOMILE : medicinal, biochemical, and agricultural aspects.*, vol. 1. Taylor & Francis Group - CRC Press, 1 ed., 2019.

[14] L. Elisabet Meza Peter and L. María Dicovskiy Riobóo, "Uso potencial de la manzanilla matricaria chamomilla l. y experiencias en Nicaragua," *Revista Ciencia y Tecnología El Higo*, vol. 10, pp. 1–8, 6 2020.

[15] Instituto Interamericano de Coperación para la Agricultura (IICA), *Plantas Medicinales y Otras Especies Utiles*. 2005.

[16] M. Sánchez, J. Gracia, and Q. Mendiola, *Guía visual de plantas nativas del Parque Nacional Los Cardones Valles CAlchiques* , vol. 0. 12 2015.

[17] PermaTree, "Bidens andicola planta medicinal de Ecuador – PermaTree," 6 2016.

[18] "Dalea coerulea planta medicinal de Ecuador – PermaTree," 6 2016.

[19] "ISO: Dalea coerulea – Repositorio Digital: Flora de la Mitad del Mundo, UETMM," 6 2022.

[20] G. Renoblaes and J. Sallés, "Plantas de interés farmacéutico," *Universidad del País Vasco*.

[21] "Propiedades de la capuchina (Tropaeolum majus) – Botanical-online," 2023.

[22] E. Fernández, V. Espinel, S. Gordillo, J. Ziarovaska, and J. Zepeda, "ESTUDIO ETNOBOTÁNICO DE PLANTAS MEDICINALES UTILIZADAS EN TRES CANTONES DE LA PROVINCIA IMBABURA, ECUADOR /," *Agrociencia*, vol. 53, pp. 797–810, 9 2019.

[23] T. Dominguez Minguez, *Vision artificial : aplicaciones practicas con OpenCV - Python*, vol. 1. Marcombo, 1 ed., 2021.

[24] A. Dinesh, S. D. Anitha Selvasofia, K. S. Datcheen, and D. Rakhesh Varshan, "Machine learning for strength evaluation of concrete structures – Critical review," *Materials Today: Proceedings*, 4 2023.

[25] J. Beunza, E. Puertas, B. Rodríguez, E. Condés, and J. Bonis, *Manual Práctico de Inteligencia Artificial En Entornos Sanitarios - Google Books*, vol. 1. España: Elsevier, 2 ed., 20.

[26] Kim-Sung Jie and Ming Liu, "Computer vision based real-time information acquisition for transport traffic," in *2005 IEEE International Conference on Information Acquisition*, pp. 164–168, IEEE, 2005.

[27] U. Padma, S. Jagadish, and M. K. Singh, "Recognition of plant's leaf infection by image processing approach," *Materials Today: Proceedings*, vol. 51, pp. 914–917, 1 2022.

[28] "What Is Image Recognition? - MATLAB & Simulink."

[29] J. Bobadilla, *Machine Learning y Deep Learning: Usando Python, Scikit y Keras - Jesús Bobadilla - Google Libros*. Colombia: Ra-ma Editorial, 621.39 ed., 2020.

[30] S. Srinivas and A. J. Young, "Machine Learning and Artificial Intelligence in Surgical Research," *Surgical Clinics of North America*, vol. 103, pp. 299–316, 4 2023.

[31] M. Soori, B. Arezoo, and R. Dastres, "Machine learning and artificial intelligence in CNC machine tools, A review," *Sustainable Manufacturing and Service Economics*, p. 100009, 1 2023.

[32] J. Mueller and L. Massaron, *Deep Learning for Dummies | John Paul Mueller, Luca Massaron | download*, vol. 1. Sons Inc, 1 ed., 4 2021.

[33] R. M. Adnan, H.-L. Dai, A. Kuriqi, O. Kisi, and M. Zounemat-Kermani, "Improving drought modeling based on new heuristic machine learning methods," *Ain Shams Engineering Journal*, p. 102168, 2 2023.

[34] M. J. d. l. Fuente Aparicio, E. S. Gonzalez Palenzuela, J. M. Zamarreno Cosme, and T. Calonge Cano, *Aplicaciones de las redes de neuronas en supervision, diagnosis y control de procesos*. Equinoccio, 1999.

[35] P. Viñuela and I. Galván, *Redes de Neuronas Artificiales Un Enfoque Práctico*. España: PERSON EDUCACIÓN S.A., 1 ed., 2004.

[36] B. Reagen, R. Adolf, P. Whatmough, G.-Y. Wei, and D. Brooks, *Deep learning for computer architects*, vol. 1. Madison: Princeton University, 1 ed., 5 2022.

[37] D. Sarkar, R. Bali, and T. Sharma, *Practical Machine Learning with Python*, vol. 1. India: Apress, 1 ed., 1 2018.

[38] F. Lubinus Badillo, C. Andrés, R. Hernández, B. Marconi Narváez, Y. Estefanía, and A. Trillos, "Redes neuronales convolucionales: un modelo de Deep Learning en imágenes diagnósticas. Revisión de tema," *Revista colombiana de radiología*, vol. 32, pp. 5591–5599, 9 2021.

[39] J. Jinez and W. Yugsi, *Implementación de un sistema para el reconocimiento y clasificación de la contaminación superficial de la laguna de Colta mediante algoritmos de inteligencia artificial*. PhD thesis, Universidad Nacional De Chimborazo, Riobamba, 12 2020.

[40] A. Yakovlev and O. Lisovychenko, "AN APPROACH FOR IMAGE ANNOTATION AUTOMATIZATION FOR ARTIFICIAL INTELLIGENCE MODELS LEARNING ," *Adaptive automatic control systems*, vol. 1, pp. 32–40, 8 2020.

[41] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 3523–3542, 7 2022.

[42] "Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library - Adrian Kaehler, Gary Bradski - Google Libros."

[43] M. Shah, A. Nandgave, A. Sahu, T. Gabhane, and N. Dable, "A REVIEW ON OBJECT DETECTION FOR BLIND USING MACHINE LEARNING," *www.irjmets.com @International Research Journal of Modernization in Engineering*, vol. 1, pp. 1–3, 12 2022.

[44] S. Gollapudi, *Learn Computer Vision Using OpenCV With Deep Learning CNNs and RNNs-Sunila Gollapudi*, vol. 1. eStudioCalamar, 1 ed., 2019.

[45] V. Lakshmanan, M. Gorner, and R. Gillard, *Practical machine learning for computer vision : end-to-end machine learning for images*, vol. 1. O'Reilly, 1 ed., 10 2021.

[46] J. Torres, *Python Deep Learning* , vol. 1. Colombia: Alpha, 1 ed., 10 20.

[47] B. Pang, E. Nijkamp, and Y. Wu, "Deep Learning With TensorFlow: A Review," *Journal of Education and Behavioral Statistics*, vol. 45, pp. 227–248, 9 2019.

[48] R. Mchaney, "TUTORIAL: TEXT ANALYTICS FOR SIMULATION WITH PYTHON," *Operational Research Society Simulation Workshop*, vol. 0, pp. 70–70, 3 2021.

[49] A. Lopez and A. L. Rojas, *Introducción a Python para Estudiantes de Ciencias*, vol. 1. Colombia: Editorial UPTC, 1 ed., 2021.

[50] Z. Aziz, D. Abdulqader, A. Sallow, and H. Omer, "Python Parallel Processing and Multiprocessing: A Rivew," *Academic Journal of Nawroz University*, vol. 10, pp. 345–354, 8 2021.

[51] E. Cervera, R. Marín, and J. Marín, "Más allá de Jupyter: usando Google Colab para la programación de robots," *Repositorio de la Universidad de Coruña*, vol. 0, pp. 662–669, 9 2022.

[52] J. Vittone and J. Cuello, *Diseñando apps para móviles - Google Books*, vol. 0. 1 ed., 7 2013.

[53] D. De Luca, *Apps HTML5 para moviles : Desarrollo de aplicaciones para smartphones y tablets basado en tecnologia web*, vol. 0. Buenos Aires: Alfaomega, 2 ed., 7 2022.

[54] A. Muhammad, *OpenCV Android Programming By Example.*, vol. 0. Packt Publishing, 0 ed., 11 2015.

[55] J. O. Lozada, "Investigación Aplicada: Definición, Propiedad Intelectual e Industria," *CienciAmérica: Revista de divulgación científica de la Universidad Tecnológica Indoamérica, ISSN-e 1390-9592, Vol. 3, Nº. 1, 2014, págs. 47-50*, vol. 3, no. 1, pp. 47–50, 2014.

[56] J. Mutinda, "Definition and Comparison of Common Research Methods - Google Books," 6 2018.

[57] S. Shivakumar, *Complete_Guide_to_Digital_Project_Manage*.

[58] R. Kundu, "Confusion Matrix: How To Use It & Interpret Results [Examples]," 9 2022.

Anexos

Algoritmo para el entrenamiento e identificación

```
#Comunicacion del entorno con Google Drive
from google.colab import drive
drive.mount('/content/drive')
#Montar las carpetas a google drive
from google.colab import drive
drive.mount("/content/gdrive")
#Creacion dos carpetas dentro de la carpeta modelo_entrenar
import os
%cd /content/gdrive/MyDrive/
!mkdir modelo_entrenar
%cd /content/gdrive/MyDrive/modelo_entrenar
!mkdir datos entrenamiento
#Instalacion de las librerias para la deteccion
!pip install tensorflow==2.8
!apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
!pip uninstall opencv-python --y
!pip uninstall opencv-contrib-python --y
!pip uninstall opencv-python-headless --y
!pip install opencv-python==4.5.4.60
!pip install opencv-contrib-python==4.5.4.60
!pip install opencv-python-headless==4.5.4.60
#Instalacion de la API de tensorflow
%cd  /content/
import os
import pathlib
# Clonamos del repositorio
if "models" in pathlib.Path.cwd().parts:
  while "models" in pathlib.Path.cwd().parts:
    os.chdir('..')
elif not pathlib.Path('models').exists():
  !git clone --depth 1 https://github.com/tensorflow/models
```

```
#Instalacion de la API de deteccion
%%bash
cd models/research/
protoc object_detection/protos/*.proto --python_out=.
cp object_detection/packages/tf2/setup.py .
python -m pip install .


#Datos para el entrenamiento. Imagenes - Archivos xml
%cd /content/gdrive/MyDrive/modelo_entrenar/datos

!unzip /content/gdrive/MyDrive/modelo_entrenar/datos/Imagenes.zip -d .
!unzip /content/gdrive/MyDrive/modelo_entrenar/datos/Anotaciones.zip -d
    .
#Creacion de las carpetas de entrenamiento "train" y validacion "
    Validation"
%cd /content/gdrive/MyDrive/modelo_entrenar/datos
!mkdir etiquetas_train etiquetas_validation
#Calculamos el 20% del total de la base de datos
DIR = '/content/gdrive/MyDrive/modelo_entrenar/datos/Anotaciones'
numero_datos = len([name for name in os.listdir(DIR) if os.path.isfile(
    os.path.join(DIR, name))])
datos_validacion = int(numero_datos*0.20)
print('numero de datos de validacion: '+str(datos_validacion))
#Dividimos de forma aleatoria con el numero de validacion
!ls Anotaciones/* | sort -R | head -820| xargs -I{} mv {}
    etiquetas_validation/
!ls Anotaciones/* | xargs -I{} mv {} etiquetas_train/
#Creamos los archivos CSV y mapa de etiqueta .pbtxt
%cd  /content/gdrive/MyDrive/modelo_entrenar/datos

import glob
import xml.etree.ElementTree as ET
import pandas as pd

def xml_to_csv(path):
  classes_names = []
  xml_list = []

  for xml_file in glob.glob(path + '/*.xml'):
    tree = ET.parse(xml_file)
    root = tree.getroot()
```

```
    for member in root.findall('object'):
      classes_names.append(member[0].text)
      value = (root.find('filename').text  ,
               int(root.find('size')[0].text),
               int(root.find('size')[1].text),
               member[0].text,
               int(member[4][0].text),
               int(member[4][1].text),
               int(member[4][2].text),
               int(member[4][3].text))
      xml_list.append(value)
  column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin',
      'xmax', 'ymax']
  xml_df = pd.DataFrame(xml_list, columns=column_name)
  classes_names = list(set(classes_names))
  classes_names.sort()
  return xml_df, classes_names

for label_path in ['etiquetas_train', 'etiquetas_validation']:
  image_path = os.path.join(os.getcwd(), label_path)
  xml_df, classes = xml_to_csv(label_path)
  xml_df.to_csv(f'{label_path}.csv', index=None)
  print(f'Se convirtio los datos {label_path} xml to csv.')

label_map_path = os.path.join("label_map.pbtxt")
pbtxt_content = ""

for i, class_name in enumerate(classes):
    pbtxt_content = (
        pbtxt_content
        + "item {{\n    id: {0}\n    name: '{1}'\n}}\n\n".format(i + 1,
           class_name)
    )
pbtxt_content = pbtxt_content.strip()
with open(label_map_path, "w") as f:
    f.write(pbtxt_content)
    print('Se creo el mapa de etiquetas label_map.pbtxt ')

#Con el archivo subido para generar los archivos tfrecord
!python /content/gdrive/MyDrive/modelo_entrenar/datos/generate_tfrecord.
   py\
        /content/gdrive/MyDrive/modelo_entrenar/datos/etiquetas_train.
```

```
            csv \
        /content/gdrive/MyDrive/modelo_entrenar/datos/label_map.pbtxt\
        /content/gdrive/MyDrive/modelo_entrenar/datos/Imagenes \
        /content/gdrive/MyDrive/modelo_entrenar/datos/train.record

!python /content/gdrive/MyDrive/modelo_entrenar/datos/generate_tfrecord.
    py\
        /content/gdrive/MyDrive/modelo_entrenar/datos/
            etiquetas_validation.csv\
        /content/gdrive/MyDrive/modelo_entrenar/datos/label_map.pbtxt\
        /content/gdrive/MyDrive/modelo_entrenar/datos/Imagenes\
        /content/gdrive/MyDrive/modelo_entrenar/datos/validation.record

#Entrenamiento

#Creamos la carpeta para el modelo pre entrenado
%cd /content/gdrive/MyDrive/modelo_entrenar/datos
!mkdir pre_modelo

#Descargamos el modelo

%cd /content/gdrive/MyDrive/modelo_entrenar/datos/pre_modelo
!wget http://download.tensorflow.org/models/object_detection/tf2
    /20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz
!tar -xzvf ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz

#Configuracion del entrenamiento a traves de archivo pipeline.config

pipeline_file = '/content/gdrive/MyDrive/modelo_entrenar/datos/
    pipeline_nuevo.config'
model_dir = '/content/gdrive/MyDrive/modelo_entrenar/entrenamiento'

#Inicia el en entrenamiento del modelo

!python /content/models/research/object_detection/model_main_tf2.py \
    --pipeline_config_path={pipeline_file} \
    --model_dir={model_dir} \
    --num_train_steps=60000

#Evaluacion del modelo

!python /content/models/research/object_detection/model_main_tf2.py \
```

```
    --pipeline_config_path={pipeline_file} \
    --model_dir={model_dir} \
    --checkpoint_dir={model_dir}\
    --alsologtostderr
#Visualizacion del modelo
%load_ext tensorboard
%tensorboard --logdir '/content/gdrive/MyDrive/modelo_entrenar/
   entrenamiento'

#Exportacion del modelo en la carpeta

%cd /content/gdrive/MyDrive/modelo_entrenar/datos
!mkdir fine_tuned_model
import re
import numpy as np
output_directory = '/content/gdrive/MyDrive/modelo_entrenar/datos/
   fine_tuned_model'
pipeline_file = '/content/gdrive/MyDrive/modelo_entrenar/datos/
   pipeline_nuevo.config'
last_model_path = '/content/gdrive/MyDrive/modelo_entrenar/entrenamiento
   '
!python /content/models/research/object_detection/exporter_main_v2.py \
    --trained_checkpoint_dir {last_model_path} \
    --output_directory {output_directory} \
    --pipeline_config_path {pipeline_file}

#Visualizamos la inferencia del modelo con una imagen de dataset

%cd /content/models/research/object_detection
import tensorflow as tf
import time
import numpy as np
import warnings
warnings.filterwarnings('ignore')
from PIL import Image
from google.colab.patches import cv2_imshow
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
IMAGE_SIZE = (12, 8) # tamano
import matplotlib.pyplot as plt
PATH_TO_SAVED_MODEL="/content/gdrive/MyDrive/modelo_entrenar/datos/
   fine_tuned_model/saved_model/"
```

```
print('Cargando el modelo...', end='')
detect_fn=tf.saved_model.load(PATH_TO_SAVED_MODEL)
print(' Listo!')

#Inferencia del modelo
import numpy as np
category_index=label_map_util.create_category_index_from_labelmap("/
    content/gdrive/MyDrive/modelo_entrenar/datos/label_map.pbtxt",
    use_display_name=True)
def load_image_into_numpy_array(path):
     return np.array(Image.open(path))
image_path = "/content/gdrive/MyDrive/modelo_entrenar/datos/Imagenes
    /1001.jpg"
image_np = load_image_into_numpy_array(image_path)
input_tensor = tf.convert_to_tensor(image_np)
input_tensor = input_tensor[tf.newaxis, ...]
detections = detect_fn(input_tensor)
num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
               for key, value in detections.items()}
detections['num_detections'] = num_detections
detections['detection_classes'] = detections['detection_classes'].astype
    (np.int64)
image_np_with_detections = image_np.copy()
viz_utils.visualize_boxes_and_labels_on_image_array(
      image_np_with_detections,
      detections['detection_boxes'],
      detections['detection_classes'],
      detections['detection_scores'],
      category_index,
      use_normalized_coordinates=True,
      max_boxes_to_draw=200,
      min_score_thresh=.5, #umbral
      agnostic_mode=False)
%matplotlib inline
plt.figure(figsize=IMAGE_SIZE, dpi=200)
plt.axis("off")
plt.imshow(image_np_with_detections)
plt.show()

#Exportacion del modelo a TensorFlow LITE
!pip install tf-nightly
```

```
!python /content/models/research/object_detection/
    export_tflite_graph_tf2.py \
--pipeline_config_path /content/gdrive/MyDrive/modelo_entrenar/datos/
    pipeline_nuevo.config \
--trained_checkpoint_dir /content/gdrive/MyDrive/modelo_entrenar/
    entrenamiento \
--output_directory /content/gdrive/MyDrive/modelo_entrenar/datos/tflite
#Guardamo el modelo en punto flotante 32
%cd /content/gdrive/MyDrive/modelo_entrenar/datos
import tensorflow as tf
saved_model_dir = '/content/gdrive/MyDrive/modelo_entrenar/datos/tflite/
    saved_model'
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
tflite_model = converter.convert()
open("/content/gdrive/MyDrive/modelo_entrenar/datos/tflite/model_fp32.
    tflite", "wb").write(tflite_model)

#Reduccion del modelo a punto flotante 16

import tensorflow as tf
import numpy as np
saved_model_dir = '/content/gdrive/MyDrive/modelo_entrenar/datos/tflite/
    saved_model/'
dataset_dir = '/content/gdrive/MyDrive/modelo_entrenar/datos/tflite'
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]
tflite_quant_model = converter.convert()
# save model
open("/content/gdrive/MyDrive/modelo_entrenar/datos/tflite/model_fp16.
    tflite", "wb").write(tflite_quant_model)

#Cargamos los metadatos en el modelo
!pip install tflite_support_nightly
%cd /content/gdrive/MyDrive/modelo_entrenar/datos/
!mkdir metadata
from tflite_support.metadata_writers import object_detector
from tflite_support.metadata_writers import writer_utils
ObjectDetectorWriter = object_detector.MetadataWriter
_MODEL_PATH = "/content/gdrive/MyDrive/modelo_entrenar/datos/tflite/
    model_fp32.tflite"
_LABEL_FILE = "/content/gdrive/MyDrive/modelo_entrenar/datos/labelmap.
```

```
    txt"
_SAVE_TO_PATH = "/content/gdrive/MyDrive/modelo_entrenar/datos/metadata/
    model_fp32_meta.tflite"
_INPUT_NORM_MEAN = 127.5
_INPUT_NORM_STD = 127.5
writer = ObjectDetectorWriter.create_for_inference(
    writer_utils.load_file(_MODEL_PATH), [_INPUT_NORM_MEAN], [
        _INPUT_NORM_STD], [_LABEL_FILE])
print(writer.get_metadata_json())
writer_utils.save_file(writer.populate(), _SAVE_TO_PATH)
```

Código para calcular la matriz de confusión del modelo de identificación de plantas.

```
#Extraemos las etiquetas reales e imagenes de la carpeta en formato .zip
%cd /content/gdrive/MyDrive/modelo_entrenar/datos/ConfusionMatrix
!unzip /content/gdrive/MyDrive/modelo_entrenar/datos/ConfusionMatrix/
    etiquetas_validation.zip -d .
!unzip /content/gdrive/MyDrive/modelo_entrenar/datos/ConfusionMatrix/
    Validation_

#Tranformamos a las etiquetas reales en un formato .cvs
import os
import glob
import csv
import xml.etree.ElementTree as ET
etiquetas_reales_folder = "/content/gdrive/MyDrive/modelo_entrenar/datos
    /Confusio
csv_etiquetas_reales = "/content/gdrive/MyDrive/modelo_entrenar/datos/
    ConfusionMatrix
etiquetas_reales_files = [f for f in glob.glob(os.path.join(
    etiquetas_reales_folder, "*.xml"))]
with open(csv_etiquetas_reales, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Imagen", "Etiquetas Reales"])
    for etiquetas_file in etiquetas_reales_files:
        tree = ET.parse(etiquetas_file)
        root = tree.getroot()
        etiquetas = [obj.find("name").text for obj in root.findall("
            object")]
        image_name = root.find("filename").text
        etiquetas_combinadas = ', '.join(etiquetas)
        writer.writerow([image_name, etiquetas_combinadas])
```

```
print("Etiquetas reales guardadas en", csv_etiquetas_reales)

#Obtenemos las etiquetas predichas por el modelo.
import os
import numpy as np
import tensorflow as tf
from PIL import Image
import glob
import csv
from object_detection.utils import label_map_util

label_map_path = "/content/gdrive/MyDrive/modelo_entrenar/datos/
    label_map.pbtxt"
model = tf.saved_model.load("/content/gdrive/MyDrive/modelo_entrenar/
    datos/fine_tuned_model/saved_model/")

def load_image_into_numpy_array(path):
    return np.array(Image.open(path))
# umbral de confianza
umbral_de_confianza = 1
category_index = label_map_util.create_category_index_from_labelmap(
    label_map_path, use_display_name=True)
folder_path = "/content/gdrive/MyDrive/modelo_entrenar/datos/
    ConfusionMatrix/Validation_Data/"
valid_image_extensions = ['.jpg', '.jpeg', '.png']
image_paths = [f for f in glob.glob(folder_path + "*") if os.path.isfile
    (f) and f.lower().endswith(tuple(valid_image_extensions))]
csv_file = "/content/gdrive/MyDrive/modelo_entrenar/datos/
    ConfusionMatrix/etiquetas_predichas.csv"
with open(csv_file, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Imagen", "Etiquetas Predichas"])
    for image_path in image_paths:
        image_np = load_image_into_numpy_array(image_path)
        input_tensor = tf.convert_to_tensor(image_np)
        input_tensor = input_tensor[tf.newaxis, ...]
        detections = model(input_tensor)
        predicted_labels = detections['detection_classes'][0].numpy().
            astype(np.int64)
        confidence_scores = detections['detection_scores'][0].numpy()
        detecciones_filtradas = [(category_index[label]['name']) for
            label, score in zip(predicted_labels, confidence_scores) if
```

```
            score >= umbral_de_confianza]
        etiquetas_predichas = ', '.join(detecciones_filtradas)
        if len(detecciones_filtradas) > 0:
            image_name = os.path.basename(image_path)
            writer.writerow([image_name, etiquetas_predichas])
print("Resultados guardados en", csv_file)

#Calculamos la matriz de confusion y las metricas de evaluacion

import csv
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

ruta_etiquetas_reales = "/content/gdrive/MyDrive/modelo_entrenar/datos/
    ConfusionMatrix/etiquetas_validation.csv"
ruta_etiquetas_predichas = "/content/gdrive/MyDrive/modelo_entrenar/
    datos/ConfusionMatrix/etiquetas_predichas.csv"

etiquetas_reales = []
etiquetas_predichas = []

with open(ruta_etiquetas_reales, 'r') as file:
    reader = csv.reader(file)
    next(reader)
    for row in reader:
        etiquetas_reales.append(row[1])
with open(ruta_etiquetas_predichas, 'r') as file:
    reader = csv.reader(file)
    next(reader)  # Salta la cabecera del archivo
    for row in reader:
        etiquetas_predichas.append(row[1])
etiquetas_reales = [etiquetas.split(', ') for etiquetas in
    etiquetas_reales]
etiquetas_predichas = [etiquetas.split(', ') for etiquetas in
    etiquetas_predichas]

etiquetas_unicas = list(set([label for sublist in etiquetas_reales +
    etiquetas_predichas for label in sublist]))

matriz_confusion = confusion_matrix([etiqueta for sublist in
    etiquetas_reales for etiqueta in sublist],
                                    [etiqueta for sublist in
```

```
                                        etiquetas_predichas for etiqueta
                                         in sublist],
                                    labels=etiquetas_unicas)


print("Matriz de Confusion:")
print(matriz_confusion)
from sklearn.metrics import classification_report
n_muestras = len(matriz_confusion)
etiquetas_clases = etiquetas_unicas
y_true = [etiqueta for sublist in etiquetas_reales for etiqueta in
    sublist]
y_pred = [etiqueta for sublist in etiquetas_predichas for etiqueta in
    sublist]

reporte = classification_report(y_true, y_pred, target_names=
    etiquetas_clases, output_dict=True)

for etiqueta in etiquetas_clases:
    print("-----------------------------------------------------")
    print(f"Clase: {etiqueta}")
    print(f"Precision: {reporte[etiqueta]['precision']:.2f}")
    print(f"Recall: {reporte[etiqueta]['recall']:.2f}")
    print(f"F1-Score: {reporte[etiqueta]['f1-score']:.2f}")


print("-----------------------------------------------------------")
print(f"Exactitud (Accuracy): {reporte['accuracy']:.2f}")
print(f"Promedio Macro (Macro Avg):")
print(f"Precision: {reporte['macro avg']['precision']:.2f}")
print(f"Recall: {reporte['macro avg']['recall']:.2f}")
print(f"F1-Score: {reporte['macro avg']['f1-score']:.2f}")
print("-----------------------------------------------------------")

print(f"Promedio Ponderado (Weighted Avg):")
print(f"Precision: {reporte['weighted avg']['precision']:.2f}")
print(f"Recall: {reporte['weighted avg']['recall']:.2f}")
print(f"F1-Score: {reporte['weighted avg']['f1-score']:.2f}")

#Grafica de la matriz de confusion mediante mapa de calor

import csv
```

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(matriz_confusion, annot=True, fmt="d", cmap="Blues",
    xticklabels=etiquetas_unicas, yticklabels=etiquetas_unicas)
plt.xlabel('Valores predichos')
plt.ylabel('Valores reales')
plt.title('Matriz de Confusion')
plt.show()
```

Desarrollo de aplicación móvil.

CameraFragment.kt

```
//Copyright 2022 The TensorFlow Authors. All Rights Reserved
package org.tensorflow.lite.examples.objectdetection.fragments

import android.annotation.SuppressLint
import android.content.res.Configuration
import android.graphics.Bitmap
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.AdapterView
import android.widget.Toast
import androidx.camera.core.AspectRatio
import androidx.camera.core.Camera
import androidx.camera.core.CameraSelector
import androidx.camera.core.ImageAnalysis
import androidx.camera.core.ImageAnalysis.OUTPUT_IMAGE_FORMAT_RGBA_8888
import androidx.camera.core.ImageProxy
import androidx.camera.core.Preview
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.core.content.ContextCompat
import androidx.fragment.app.Fragment
import androidx.navigation.Navigation
import java.util.LinkedList
import java.util.concurrent.ExecutorService
```

```
import java.util.concurrent.Executors
import org.tensorflow.lite.examples.objectdetection.ObjectDetectorHelper
import org.tensorflow.lite.examples.objectdetection.R

class CameraFragment : Fragment(), ObjectDetectorHelper.DetectorListener
    {
    companion object{
    private val TAG = "ObjectDetection" }
    private var _fragmentCameraBinding: FragmentCameraBinding? = null
        get() = _fragmentCameraBinding!!
    private lateinit var objectDetectorHelper: ObjectDetectorHelper
    private lateinit var bitmapBuffer: Bitmap
    private var preview: Preview? = null
    private var imageAnalyzer: ImageAnalysis? = null
    private var camera: Camera? = null
    private var cameraProvider: ProcessCameraProvider? = null
    private lateinit var cameraExecutor: ExecutorService
    override fun onResume() {
        super.onResume()
        if (!PermissionsFragment.hasPermissions(requireContext())) {
            Navigation.findNavController(requireActivity(), R.id.
                fragment_container)
                .navigate(CameraFragmentDirections.
                    actionCameraToPermissions())
        }
    }
    override fun onDestroyView() {
        _fragmentCameraBinding = null
        super.onDestroyView()
        cameraExecutor.shutdown()
    }
    override fun onCreateView(
      inflater: LayoutInflater,
      container: ViewGroup?,
      savedInstanceState: Bundle?
    ): View {
        _fragmentCameraBinding = FragmentCameraBinding.inflate(inflater,
            container, false)
        return fragmentCameraBinding.root
    }
    @SuppressLint("MissingPermission")
    override fun onViewCreated(view: View, savedInstanceState: Bundle?)
```

```
        {
        super.onViewCreated(view, savedInstanceState)
        objectDetectorHelper = ObjectDetectorHelper(
            context = requireContext(),
            objectDetectorListener = this)

        // Initialize our background executor
        cameraExecutor = Executors.newSingleThreadExecutor()
        // Wait for the views to be properly laid out
        fragmentCameraBinding.viewFinder.post {
            setUpCamera()
        }
        initBottomSheetControls()
    }
    private fun initBottomSheetControls() {
        // When clicked, lower detection score threshold floor
        fragmentCameraBinding.bottomSheetLayout.thresholdMinus.
            setOnClickListener {
            if (objectDetectorHelper.threshold >= 0.05) {
                objectDetectorHelper.threshold -= 0.05f
                updateControlsUi()
            }
        }
        // When clicked, raise detection score threshold floor
        fragmentCameraBinding.bottomSheetLayout.thresholdPlus.
            setOnClickListener {
            if (objectDetectorHelper.threshold <= 1) {
                objectDetectorHelper.threshold += 0.05f
                updateControlsUi()
            }
        }
        fragmentCameraBinding.bottomSheetLayout.maxResultsMinus.
            setOnClickListener {
            if (objectDetectorHelper.maxResults > 1) {
                objectDetectorHelper.maxResults--
                updateControlsUi()
            }
        }
        fragmentCameraBinding.bottomSheetLayout.maxResultsPlus.
            setOnClickListener {
            if (objectDetectorHelper.maxResults < 5) {
                objectDetectorHelper.maxResults++
```

```
                updateControlsUi()
            }
        }
        fragmentCameraBinding.bottomSheetLayout.threadsMinus.
            setOnClickListener {
            if (objectDetectorHelper.numThreads > 1) {
                objectDetectorHelper.numThreads--
                updateControlsUi()
            }
        }
        fragmentCameraBinding.bottomSheetLayout.threadsPlus.
            setOnClickListener {
            if (objectDetectorHelper.numThreads < 4) {
                objectDetectorHelper.numThreads++
                updateControlsUi()
            }
        }
        fragmentCameraBinding.bottomSheetLayout.spinnerDelegate.
            setSelection(0, false)
        fragmentCameraBinding.bottomSheetLayout.spinnerDelegate.
            onItemSelectedListener =
            object : AdapterView.OnItemSelectedListener {
                override fun onItemSelected(p0: AdapterView<*>?, p1: p2:
                    Int, p3: Long) {
                    objectDetectorHelper.currentDelegate = p2
                    updateControlsUi()
                }
                override fun onNothingSelected(p0: AdapterView<*>?) {
                    /* no op */
                }
            }
        fragmentCameraBinding.bottomSheetLayout.spinnerModel.
            setSelection(0, false)
        fragmentCameraBinding.bottomSheetLayout.spinnerModel.
            onItemSelectedListener =
            object : AdapterView.OnItemSelectedListener {
                override fun onItemSelected(p0: AdapterView<*>?, p1:
                   View?, p2: Int, p3: Long) {
                    objectDetectorHelper.currentModel = p2
                    updateControlsUi()
                }
                override fun onNothingSelected(p0: AdapterView<*>?) {
```

```
                    /* no op */
                }
            }
        }
        private fun updateControlsUi() {
            fragmentCameraBinding.bottomSheetLayout.maxResultsValue.text =
                objectDetectorHelper.maxResults.toString()
            fragmentCameraBinding.bottomSheetLayout.thresholdValue.text =
                String.format("%.2f", objectDetectorHelper.threshold)
            fragmentCameraBinding.bottomSheetLayout.threadsValue.text =
                objectDetectorHelper.numThreads.toString()
            objectDetectorHelper.clearObjectDetector()
            fragmentCameraBinding.overlay.clear()
        }
        private fun setUpCamera() {
            val cameraProviderFuture = ProcessCameraProvder.getInstance(
               requireContext())
            cameraProviderFuture.addListener(
                {
                    // CameraProvider
                    cameraProvider = cameraProviderFuture.get()
                    // Build and bind the camera use cases
                    bindCameraUseCases()
                },
                ContextCompat.getMainExector(requireContext())
            )
        }
        @SuppressLint("UnsafeOptInUsageError")
        private fun bindCameraUseCases() {
            // CameraProvider
            val cameraProvider =
                cameraProvider ?: throw IllegalStateException("Camera
                    initialization failed.")
            val cameraSelector =
                CameraSelector.Builder().requireLensFacing(CameraSelector.
                    LENS_FACING_BACK).build()
            preview =
                Preview.Builder()
                    .setTargetAspectRatio(AspectRatio.RATIO_4_3)
                    .setTargetRotation(fragmentCameraBinding.viewFinder.
                        display.rotation)
                    .build()
```

```
        imageAnalyzer =
            ImageAnalysis.Builder()
                .setTargetAspectRatio(AspectRatio.RATIO_4_3)
                .setTargetRotation(fragmentCameraBinding.viewFinder.
                    display.rotation)
                .setBackpressureStrategy(ImageAnalysis.
                    STRATEGY_KEEP_ONLY_LATEST)
                .setOutputImageFormat(OUTPUT_IMAGE_FORMAT_RGBA_8888)
                .build()
                // The analyzer can then be assigned to the instance
                .also {
                    it.setAnalyzer(cameraExecutor) { image ->
                        if (!::bitmapBuffer.isInitialized) {
                            // The image rotation and RGB image buffer
                                are initialized only once
                            // the analyzer has started running
                            bitmapBuffer = Bitmap.createBitmap(
                              image.width,
                              image.height,
                              Bitmap.Config.ARGB_8888
                            )
                        }

                        detectObjects(image)
                    }
                }

        cameraProvider.unbindAll()
        try {
            camera = cameraProvider.bindToLifecycle(this, cameraSelector
                , preview, imageAnalyzer)
            preview?.setSurfaceProvider(fragmentCameraBinding.viewFinder
                .surfaceProvider)
        } catch (exc: Exception) {
            Log.e(TAG, "Use case binding failed", exc)
        }
    }
    private fun detectObjects(image: ImageProxy) {
        image.use { bitmapBuffer.copyPixelsFromBuffer(image.planes[0].
            buffer) }
        val imageRotation = image.imageInfo.rotationDegrees
        objectDetectorHelper.detect(bitmapBuffer, imageRotation)
```

```
    }
    override fun onConfigurationChanged(newConfig: Configuration) {
        super.onConfigurationChanged(newConfig)
        imageAnalyzer?.targetRotation = fragmentCameraBinding.viewFinder
            .display.rotation
    }
    override fun onResults(
      results: MutableList<Detection>?,
      inferenceTime: Long,
      imageHeight: Int,
      imageWidth: Int
    ) {
        activity?.runOnUiThread {
            fragmentCameraBinding.bottomSheetLayout.inferenceTimeVal.
                text =
                            String.format("%d ms", inferenceTime)
            fragmentCameraBinding.overlay.setResults(
                results ?: LinkedList<Detection>(),
                imageHeight,
                imageWidth
            )
            fragmentCameraBinding.overlay.invalidate()
        }
    }
    override fun onError(error: String) {
        activity?.runOnUiThread {
            Toast.makeText(requireContext(), error, Toast.LENGTH_SHORT).
                show()
        }
    }
}
```

MainActivity.kt

```
//Copyright 2022 The TensorFlow Authors. All Rights Reserved
package org.tensorflow.lite.examples.objectdetection
import android.content.Intent
import android.os.Build
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
```

```
import org.tensorflow.lite.examples.objectdetection.databinding.
    ActivityMainBinding
import org.tensorflow.lite.examples.objectdetection.fragments.Properties

class MainActivity : AppCompatActivity() {
    private lateinit var activityMainBinding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        activityMainBinding = ActivityMainBinding.inflate(layoutInflater
            )
        setContentView(activityMainBinding.root)
        initProperties()
    }
    fun initProperties() {
        val button = findViewById<Button>(R.id.ObtenerPropiedades)
        button.setOnClickListener {
            val specie = findViewById<TextView>(R.id.Especie).text.
                toString()  //Get the species identify form TextView
            val intent = Intent(this, Properties::class.java)
            intent.putExtra("Especie",specie)
            startActivity(intent)
        }
    }
    override fun onBackPressed() {
        if (Build.VERSION.SDK_INT == Build.VERSION_CODES.Q) {
            // Workaround for Android Q memory leak issue in
                IRequestFinishCallback$Stub.
            // (https://issuetracker.google.com/issues/139738913)
            finishAfterTransition()
        } else {
            super.onBackPressed()
        }
    }
}
```

ObjectDetectorHelper.kt

```
//Copyright 2022 The TensorFlow Authors. All Rights Reserved
package org.tensorflow.lite.examples.objectdetection

import android.content.Intent
import android.os.Build
```

```
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import org.tensorflow.lite.examples.objectdetection.databinding.
   ActivityMainBinding
import org.tensorflow.lite.examples.objectdetection.fragments.Properties

class MainActivity : AppCompatActivity() {

    private lateinit var activityMainBinding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle? {
        super.onCreate(savedInstanceState)
        activityMainBinding = ActivityMainBinding.inflate(layoutInflater
            )
        setContentVie(activityMainBinding.root)
        initProperties()

    }
    fun initProperties() {
        val button = findViewById<Button>(R.id.ObtenerPropiedades)
        button.setOnClickListener {
            val specie = findViewById<TextView>(R.id.Especie).text.
                toString()  //Get the species identify form TextView
            val intent = Intent(this, Properties::class.java)
            intent.putExtr("Especie",specie)
            startActivity(intent)

        }
    }
    override fun onBackPressed() {
        if (Build.VERSION.SDK_INT == Build.VERSION_CODES.Q) {
            // Workaround for Android Q memory leak issue in
                IRequestFinishCallback$Stub.
            // (https://issuetracker.google.com/issues/139738913)
            finishAfterransition()
        } else {
            super.onBackPressed()
        }
    }
}
```

OverlayView.kt

```
//Copyright 2022 The TensorFlow Authors. All Rights Reserved
package org.tensorflow.lite.examples.objectdetection
import android.app.Activity
import android.content.Context
import android.graphics.Canvas
import android.graphics.Color
import android.graphics.Paint
import android.graphics.Rect
import android.graphics.RectF
import android.util.AttributeSet
import android.view.View
import android.widget.TextView
import androidx.core.content.ContextCompat
import org.tensorflow.lite.examples.objectdetection.fragments.Properties
import java.util.LinkedList
import kotlin.math.max
import org.tensorflow.lite.task.vision.detector.Detection

class  OverlayView(context: Context?, attrs: AttributeSet?) : View(
    context, attrs) {
    private var results: List<Detection> = LinkedList<Detection>()
    private var boxPaint = Paint()
    private var textBackgroundPaint = Paint()
    private var textPaint = Paint()
    private var scaleFactor: Float = 1f
    private var bounds = Rect()
    private var especie: TextView?= null
    private var Porcentaje: TextView?= null
    init {
        initPaints()
        especie=(context as? Activity)?.findViewById(R.id.Especie)
        Porcentaje=(context as? Activity)?.findViewById(R.id.Porcentaje)
    }
    fun clear() {
        textPaint.reset()
        textBackgroundPaint.reset()
        boxPaint.reset()
        invalidate()
        initPaints()
    }
```

```
    private fun initPaints() {
        textBackgroundPaint.color = Color.BLACK
        textBackgroundPaint.style = Paint.Style.FILL
        textBackgroundPaint.textSize = 50f
        textPaint.color = Color.WHITE
        textPaint.style = Paint.Style.FILL
        textPaint.textSize = 50f
        boxPaint.color = ContextCompat.getColor(context!!, R.color.
            bounding_box_color)
        boxPaint.strokeWidth = 8F
        boxPaint.style = Paint.Style.STROKE
    }
    override fun draw(canvas: Canvas) {
        super.draw(canvas)
        for (result in results) {
            val boundingBox = result.boundingBox
            val top = boundingBox.top * scaleFactor
            val bottom = boundingBox.bottom * scaleFactor
            val left = boundingBox.left * scaleFactor
            val right = boundingBox.right * scaleFactor

            val drawableRect = RectF(left, top, right, bottom)
            canvas.drawRect(drawableRect, boxPaint)

            val labely=result.categories[0].label
            val percentage= String.format("%.2f%%", result.categories
                [0].score*100 )
            val drawableText =
                labely + " " + percentage
            println(" $drawableText")
            especie?.text=labely
            Porcentaje?.text=percentage           textBackgroundPaint.
                getTextBounds(drawableText, 0, drawableText.length,
                bounds)
            val textWidth = bounds.width()
            val textHeight = bounds.height()
            canvas.drawRect(
                left,
                top,
                left + textWidth + Companion.BOUNDING_RECT_TEXT_PADDING,
                top + textHeight + Companion.BOUNDING_RECT_TEXT_PADDING,
                textBackgroundPaint
```

```
            )
            canvas.drawText(drawableText, left, top + bounds.height(),
                textPaint)
        }
    }
    fun setResults(
      detectonResults: MutableList<Detection>,
      imageHeight: Int,
      imageWidth: Int,
    ) {
        results = detectionResults
        scalFactor = max(width * 1f / imageWidth, height * 1f /
            imageHeight)
    }
    companion object {
        private const val BOUNDING_RECT_TEXT_PADDING = 8
    }
}
```

Properties.kt

```
package org.tensorflow.lite.examples.objectdetection.fragments

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.widget.ImageView
import android.widget.TextView
import org.tensorflow.lite.examples.objectdetection.R
class Properties : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_properties)
        characteristicHeigh()
        characteristicFlower()
        characteristicHabit()
        characteristicType()
        characteristicName()
        otherName()
        descriptions()
        pain_properties()
```

```
        pain1_properties()
        Respiratoio()
        Piel()
        Renal()
        Neuro()
        antiflamatorio()
        Fiebre()
        Muscular()
        Usos()
        sintomasinfu()
        info_infu()
        sintomastopico()
        uso_topico()
        imageFlower()

    }
    fun characteristicHeigh() {
        val altura = findViewById<TextView>(R.id.altura)
        val especieText = intent.getStringExtra("Especie")?.trim()
        Log.d("Informacion util", "Especie: $especieText")
        when (especieText) {
            "manzanilla" -> {
                altura.text = getString(R.string.characteristicHeigh_m)
            }
            "mastuerzo" -> {
                altura.text = getString(R.string.characteristicHeigh_ma)
            }
            "Iso" -> {
                altura.text = getString(R.string.characteristicHeigh_i)
            }
            "Niachag" -> {
                altura.text = getString(R.string.characteristicHeigh_n)
            }
            else -> {
                altura.text = " "
            }
        }

    }
    fun characteristicFlower() {
        val flor = findViewById<TextView>(R.id.flor)
        val especieText = intent.getStringExtra("Especie")?.trim()
```

```
        when (especieText) {
            "manzanilla" -> {
                flor.text = getString(R.string.characteristicFlower_m)
            }
            "mastuerzo" -> {
                flor.text = getString(R.string.characterstic Flower_ma)
            }
            "Iso" -> {
                flor.text = getString(R.string.characteristicFlower_i)
            }
            "Niachag" -> {
                flor.text = getString(R.string.characteristicFlower_n)
            }
            else -> {
                flor.text = " "
            }
        }
    }
    fun characteristicHabit() {
        val habitat = findViewById<TextView>(R.id.habitat)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                habitat.text = getString(R.string.characteristicH_m)
            }
            "mastuerzo" -> {
                habitat.text = getString(R.string.characteristicH_ma)
            }
            "Iso" -> {
                habitat.text = getString(R.string.characteristicH_i)
            }
            "Niachag" -> {
                habitat.text = getString(R.string.characteristicH_n)
            }
            else -> {
                habitat.text = " "
            }
        }
    }
    fun characteristicType() {
        val tipo = findViewById<TextView>(R.id.type)
        val especieText = intent.getStringExtra("Especie")?.trim()
```

```
        when (especieText) {
            "manzanilla" -> {
                tipo.text = getString(R.string.characteristicT_m)
            }
            "mastuerzo" -> {
                tipo.text = getString(R.string.characteristicT_ma)
            }
            "Iso" -> {
                tipo.text = getString(R.string.characteristicT_i)
            }
            "Niachag" -> {
                tipo.text = getString(R.string.characteristicT_n)
            }
            else -> {
                tipo.text = " "
            }
        }
    }
    fun characteristicName() {
        val cientific = findViewById<TextView>(R.id.cientific)
        val especiText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                cientific.text = getString(R.string.characteristicN_m)
            }
            "mastuerzo" -> {
                cientific.text = getString(R.string.characteristicN_ma)
            }
            "Iso" -> {
                cientific.text = getString(R.string.characteristicN_i)
            }
            "Niachag" -> {
                cientific.text = getStrng(R.string.characteristicN_n)
            }
            else -> {
                cientific.text = " "
            }
        }
    }
    fun otherName() {
        val newname = findViewById<TextView>(R.id.newname)
        val especieText = intent.getStringExtra("Especie")?.trim()
```

```
        when (especieText) {
            "manzanilla" -> {
                newname.text = getString(R.string.Othername_m)
            }
            "mastuerzo" -> {
                newname.text = getString(R.string.Othername_ma)
            }
            "Iso" -> {
                newname.text = getString(R.string.Othername_i)
            }
            "Niachag" -> {
                newname.text = getString(R.string.Othername_n)
            }
            else -> {
                newname.text = " "
            }
        }
    }
    fun descriptions() {
        val descriptions = findViewById<TextView>(R.id.newdescription)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                descriptions.text = getString(R.string.descriptions_m)
            }
            "mastuerzo" -> {
                descriptions.text = getString(R.string.descriptions_ma)
            }
            "Iso" -> {
                descriptions.txt = getString(R.string.descriptions_i)
            }
            "Niachag" -> {
                descriptions.text = getString(R.string.descriptions_n)
            }
            else -> {
                descriptions.text = " "
            }
        }
    }
    fun pain_properties() {
        val pain = findViewById<TextView>(R.id.pain)
        val especieText = intent.getStringExtra("Especie")?.trim()
```

```
        when (especieText) {
            "manzanilla" -> {
                pain.text = getString(R.string.check_properties)
            }
            "mastuerzo" -> {
                pain.text = getString(R.string.fail_properties)
            }
            "Iso" -> {
                pain.tet = getString(R.string.check_properties)
            }
            "Niachag" -> {
                pain.text = getString(R.string.check_properties)
            }
            else -> {
                pain.text = " "
            }
        }
    }
    fun pain1_properties() {
        val pain1 = findViewById<TextView>(R.id.infection)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                pain1.text = getString(R.string.check_properties)
            }
            "mastuerzo" -> {
                pain1.text = getString(R.string.fail_properties)
            }
            "Iso" -> {
                pain1.text = getString(R.string.check_properties)
            }
            "Niachag" -> {
                pain1.txt = getString(R.string.fail_properties)
            }
            else -> {
                pain1.text = " "
            }
        }
    }
    fun Respiratoio() {
        val respir = findViewById<TextView>(R.id.respirar)
        val especieText = intent.getStringExtra("Especie")?.trim()
```

```
        when (especieText) {
            "manzanilla" -> {
                respir.text = getString(R.string.check_properties)
            }
            "mastuerzo" -> {
                respir.text = getString(R.string.check_properties)
            }
            "Iso" -> {
                respir.text = getString(R.string.check_properties)
            }
            else -> {
                respir.text = ""
            }
        }
    }
    fun Piel() {
        val Piell = findViewById<TextView>(R.id.piel1)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                Piell.text = getString(R.string.check_properties)
            }
            "mastuerzo" -> {
                Piell.text = getString(R.string.check_properties)
            }
            "Iso" -> {
                Piell.text = getString(R.string.check_properties)
            }
            "Niachag" -> {
                Piell.text = getString(R.string.fail_properties)
            }
            else -> {
                Piell.text = ""
            }
        }
    }
    fun Renal() {
        val Uro = findViewById<TextView>(R.id.Urologico)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                Uro.text = getString(R.string.fail_properties)
```

```
            }
            "mastuerzo" -> {
                Uro.text = getString(R.string.check_properties)
            }
            "Niachag" -> {
                Uro.text = getString(R.string.fail_properties)
            }
            else -> {
                Uro.text = ""
            }
        }
    }
    fun Neuro() {
        val Neurolo = findViewById<TextView>(R.id.Neurologico)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                Neurolo.text = getString(R.string.fail_properties)
            }
            "mastuerzo" -> {
                Neurolo.text = getString(R.string.fail_properties)
            }
            "Iso" -> {
                Neurolo.text = getString(R.string.check_properties)
            }
            "Niachag" -> {
                Neurolo.text = getString(R.string.fail_properties)
            }
            else -> {
                Neurolo.text = ""
            }
        }
    }
    fun antiflamatorio() {
        val inflamator = findViewById<TextView>(R.id.Inflamatorio)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                inflamator.text = getString(R.string.check_properties)
            }
            "mastuerzo" -> {
                inflamator.text = getString(R.string.check_properties)
```

```
            }
            "Iso" -> {
                inflamator.text = getString(R.string.check_properties)
            }
            "Niachag" -> {
                inflamator.text = getString(R.string.fail_properties)
            }
            else -> {
                inflamator.text = ""
            }
        }
    }
    fun Fiebre() {
        val calor = findViewById<TextView>(R.id.fiebre)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                calor.text = getString(R.string.check_properties)
            }
            "mastuerzo" -> {
                calor.text = getString(R.string.fail_properties)
            }
            "Iso" -> {
                calor.text = getString(R.string.fail_properties)
            }
            "Niachag" -> {
                calor.text = getString(R.string.check_properties)
            }
            else -> {
                calor.text = ""
            }
        }
    }
    fun Muscular() {
        val msco = findViewById<TextView>(R.id.musculo)
        val especieText = intent.getStringExtra("Especie")?.trim(
        when (especieText) {
            "manzanilla" -> {
                musco.text = getString(R.string.fail_properties)
            }
            "mastuerzo" -> {
                musco.text = getString(R.string.fail_properties)
```

```
            }
            "Iso" -> {
                musco.text = getString(R.string.fail_properties)
            }
            "Niachag" -> {
                muco.text = getString(R.string.check_properties)
            }
            else -> {
                musco.text = ""
            }
        }
    }
    fun Usos() {
        val use = findViewById<TextView>(R.id.part)
        val especieText = intent.getStringExtra("Especie")?.trim()

        when (especieText) {
            "manzanilla" -> {
                use.text = getString(R.string.use_m)
            }
            "masuerzo" -> {
                use.text = getString(R.string.use_ma)
            }
            "Niachag" -> {
                use.text = getString(R.string.use_n)
            }
            else -> {
                use.text = ""
            }
        }
    }
    fun sintomasinfu() {
        val numbersin = findViewById<TextView>(R.id.info2)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                numbersin.text = getString(R.string.infu_m)
            }
            "mastuerzo" -> {
                numbersin.text = getString(R.string.infu_ma)
            }
            "Iso" -> {
```

```
                numbersin.text = getString(R.string.infu_i)
            }
            "Niachag" -> {
                numbersin.text = getString(R.string.infu_n)
            }
            else -> {
                numbersin.text = "No dispone"
            }
        }
    }
    fun sintomastopico() {
        val numbertopic = findViewById<TextView>(R.id.info3)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                numbertopic.text = getString(R.string.topic_m)
            }
            "mastuerzo" -> {
                numbertopic.text = getString(R.string.topic_ma)
            }

            "Iso" -> {
                numbertopic.text = getString(R.string.topic_i)
            }
            "Niachag" -> {
                numbertopic.text = getString(R.string.topic_n)
            }
            else -> {
                numbertopic.text = "No dispone"
            }
        }
    }
    fun info_infu() {
        val info_infu = findViewById<TextView>(R.id.inf_infusion)
        val especieText = intent.getStringExtra("Especie")?.trim()
        when (especieText) {
            "manzanilla" -> {
                info_infu.text = getString(R.string.info_infu_m)
            }
            "mastuerzo" -> {
                info_infu.text = getString(R.string.info_infu_ma)
            }
```

```
            "Iso" -> {
                info_infu.text = getString(R.string.info_infu_i)
            }
            "Niachag" -> {
                info_infu.text = getString(R.string.info_infu_n)
            }
            else -> {
                info_infu.text = "No se encuentran propiedades ni usos"
            }
        }
    }
    fun uso_topico() {
            val topico1 = findViewById<TextView>(R.id.inf_topico)
            val especieText = intent.getStringExtra("Especie")?.trim()
            when (especieText) {
                "manzanilla" -> {
                    topico1.text =getString(R.string.info_topic_m)
                }
                "mastuerzo" -> {
                    topico1.text =getString(R.string.info_topic_ma)
                }
                "Iso" -> {
                    topico1.text = getString(R.string.info_topic_i)
                }
                "Niachag" -> {
                    topico1.text =getString(R.string.info_topic_n)
                }
                else -> {
                    topico1.text = ""
                }
            }
    }
    fun imageFlower() {
        val flor = findViewById<ImageView>(R.id.imageflor)
        val especieText = intent.getStringExtra("Especie")?.trim()
           when (especieText) {
               "manzanilla" -> {
                   flor.setImageResource(R.drawable.manza)
               }
               "Niachag" -> {
                   flor.setImageResource(R.drawable.na)
               }
```

```
                "mastuerzo" -> {
                    flor.setImageResource(R.drawable.ma)
                }
                "Iso" -> {
                    flor.setImageResource(R.drawable.iso)
                }
            else -> {
                flor.setImageResource(R.drawable.no)
            }
        }
    }
}
```

Activity_main.xml

```
//Copyright 2022 The TensorFlow Authors. All Rights Reserved
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:background="@android:color/transparent"
    android:layout_height="match_parent">
    <RelativeLayout
        android:id="@+id/relativeLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <androidx.fragment.app.FragmentContainerView
            android:id="@+id/fragment_container"
            android:name="androidx.navigation.fragment.NavHostFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginTop="?android:attr/actionBarSize"
            android:keepScreenOn="true"
            app:defaultNavHost="true"
            app:navGraph="@navigation/nav_graph"
            tools:context=".MainActivity" />
        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
```

```
            android:layout_alignParentTop="true"
            android:background="@color/fondoLogo">
            <ImageView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:src="@drawable/tfl_logo" />
        </androidx.appcompat.widget.Toolbar>
        <androidx.appcompat.widget.Toolbar
            android:layout_width="match_parent"
            android:layout_marginTop="55dp"
            android:background="@color/Etiqueta_Propiedades" />
        <androidx.appcompat.widget.LinearLayoutCompat
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="55dp"
            android:orientation="horizontal">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerVertical="true"
                android:textStyle="bold"
                android:text="@string/medicinal_plant_detections"></
                    TextView>
            <TextView
                android:id="@+id/Especie"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_centerVertical="true"
                android:textStyle="bold"
                android:textSize="@dimen/bottom_specie_text_size"></
                    TextView>
        </androidx.appcompat.widget.LinearLayoutCompat>
        <androidx.appcompat.widget.LinearLayoutCompat
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="80dp"
            android:orientation="horizontal">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerVertical="true"
                android:textStyle="bold"
```

```
                android:text="@string/percentage_plant_detection"></
                    TextView>
            <TextView
                android:id="@+id/Porcentaje"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_centerVertical="true"
                android:textStyle="bold"
                android:textSize="@dimen/bottom_specie_text_size"></
                    TextView>
        </androidx.appcompat.widget.LinearLayoutCompat>
        <Button
            android:id="@+id/ObtenerPropiedades"
            android:layout_width="400dp"
            android:layout_height="45dp"
            android:layout_marginTop="113dp"
            android:layout_marginStart="6dp"
            android:background="@drawable/rounded_button"
            android:text="Obtener Propiedades" />
    </RelativeLayout>
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```