Entrenamiento de redes neuronales para la identificación de plagas en cultivos de café.

Trabajo de grado:

Presentado por:

Gustavo Adolfo Avila Perez

Director:

David Alejandro Martínez Vásquez, Ph.D.

UNIVERSIDAD PEDAGÓGICA NACIONAL

FACULTAD DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN ELECTRÓNICA

BOGOTÁ D.C.

2022

Tabla de contenido

Resumen:

En este documento se encuentra el desarrollo de una aplicación para Android usando redes neuronales por medio de la librería de código abierto TensorFlow con el fin de identificar plagas en las hojas del café. Para lograr el entrenamiento de la red neuronal, se han tomado como referencia bases de datos con imágenes de plagas de Cercospora, Phoma, Roya Minador de hojas y Arañita roja para así lograr la identificación. El proceso de programación para el entrenamiento y la identificación de patrones se hizo en lenguaje Python, usando los entornos de desarrollo anaconda y Jupyter Notebook. Por otra parte, la aplicación de la red en Android se desarrolló en Java, usando el entrono Android Studio, en el cual se importó la librería TensorFlow Lite.

Palabras claves:

Aprendizaje de máquina, Procesamiento de imagen, Identificación plagas, Python, Redes neuronales.

Capítulo 1

1.1 Introducción

Las nuevas tendencias en la informática y en el análisis de datos han llevado a implementar nuevas formas de resolver problemas de identificación y clasificación de imágenes, las cuales serían complicadas mediante el uso de algoritmos clásicos de programación estructurada o incluso orientados a objetos. Por ello, desde los años 90 se ha venido trabajando en redes neuronales que buscan crear algoritmos que de forma automática puedan realizar el trabajo de identificación y clasificación a partir del uso de bases de datos que puede contener datos del tipo entero, imágenes, vídeos y sonido;

Usualmente, a esta base de datos se la conoce como set de datos de entrenamiento, y permite clasificar nuevas entradas que cumplan ciertos patrones o características.

En el caso de las imágenes, éstas deben ser agrupables en categorías que permitan identificar sus características con el fin de crear un modelo neuronal que las tome como datos de aprendizaje para que las nuevas entradas puedan ser identificadas y clasificadas, por ejemplo, desde el contexto de plagas de café, un grupo de imágenes que contienen broca servirán para saber si una nueva imagen tomada puede clasificarse dentro de este grupo o no. Los modelos neuronales que se pueden implementar son bastante diversos y abarcan un amplio espectro de funcionalidades, empezando por la simple neurona la cual tiene un comportamiento equivalente a la ecuación de la recta, pasando por modelos densos que son la unión de varias neuronas interconectadas en capas que se van activando a medida que se identifica algún valor de importancia. Por otra parte, están los modelos neuronales convolucionales, los cuales realizan operaciones matemáticas más complejas y permiten la identificación de ciertos patrones globales en las imágenes, tales como como la arquitectura ResNet50 que se ha implementado en Matlab.

Hay diferentes formas de programar las neuronas en lenguajes de programación, cómo: Matlab, Python o Java entre otros. Esto es posible debido a que la creación de la neurona puede ser un proceso sencillo y alcanzable, mediante el escalamiento de modelos neuronales de diferentes tamaños para lograr altas complejidades usando programación por objetos, y gracias a librerías como TensorFlow y Keras, mediante las

cuales se pueden implementar modelos neuronales de mayor complejidad a través de instrucciones simples, permitiendo que los modelos allí creados puedan ser exportados ya sea a otros dispositivos u otros sistemas que soporten estas dos librerías.

La librería TensorFlow que está soportada para diferentes lenguajes también está soportada para dispositivos móviles a través de su versión TensorFlow Lite, esta permite la ejecución de modelos en sistemas operativos como Android, iOS o incluso microprocesadores. Para el caso de este trabajo, se usa esta librería en una aplicación Android desarrollada en Java y compilada a través de Android Studio.

El modelo neuronal ha sido creado a partir de la recopilación de bases de datos disponibles en la web y que son de uso libre, probando diferentes tipos de redes neuronales sencillas que permiten identificar las diferencias entre estos al momento de evaluar los entrenamientos desde las pérdidas y la precisión. Este proceso de entrenamiento se realiza a través del lenguaje Python el cual es de uso libre y que puede ser ejecutado en diferentes tipos de editores ya sea Jupyter, Colab, Spyder, entre otras.

Las imágenes que se han considerado para realizar el modelo neuronal son imágenes de hojas de café que presentan, además de hojas sanas, plagas que son identificables como Cercospora, Phoma, Roya, minador de hojas y Arañita roja.

Este documento se divide en cinco capítulos. En el primer capítulo se plantea el estado del arte; en el segundo capítulo se aborda el marco teórico y el marco de implementación;

En el tercer capítulo se aborda la metodología; en el cuarto el proceso de desarrollo y resultados en cuatro momentos, y en el quinto capítulo se muestran las conclusiones, recomendaciones y trabajos a futuro.

1.2 Planteamiento del problema

En la agricultura, uno de los factores de mayor riesgo es el mal manejo de las plagas, lo cual es comúnmente contrarrestado con la aplicación de insecticidas. No obstante, la fumigación no siempre es la mejor solución dentro de las buenas prácticas agrícolas o en cultivos orgánicos, dado que el exceso de los plaguicidas provoca afectaciones en la naturaleza ya sean los sistemas bióticos (plantas y animales) o abióticos (suelo aire y agua), creando así un riesgo tanto para el medio ambiente como para las comunidades que tengan acceso directo o indirecto a estos ambientes, es decir, productores y consumidores, puesto que a medida que se va concentrando la cantidad de plaguicidas, aumentan los problemas de salud pública o las afectaciones a poblaciones de insectos que son necesarios para el cultivo en los procesos de polinización. (Del Puerto Rodríguez, Suárez Tamayo, & Palacio Estrada, 2104)

Las nuevas tendencias mundiales basadas en agricultura inteligente y de precisión, exigen que en las prácticas agrícolas se eliminen o al menos se reduzca el uso de plaguicidas y herbicidas de origen químico que pueden afectar la salud humana y a varias especies de insectos que no constituyen un riesgo para los cultivos, sino que por el contrario, son necesarios para que estos sean más sanos y productivos. En consecuencia, entre las medidas que suelen aplicarse en el este ámbito, pueden mencionarse:

- La introducción de insectos que se alimenten de plagas.
- La aplicación de plaguicidas de origen orgánico, como el tabaco, el ají y la cebolla, que no tengan un efecto residual, y que afecten solo determinadas poblaciones de insectos.
- La aplicación de insecticidas en determinados periodos, dependiendo de las condiciones climáticas, la presencia y el crecimiento de plagas (bravo y herrera, 2013).

En términos de producción, es necesario tener un control eficiente de la identificación y manejo de las diferentes plagas, puesto que, al no ser tratadas adecuadamente, pueden producir pérdidas que perjudican enormemente a los productores. Un ejemplo de ello es el reportado por el ICA en 1998, en el cual una plaga conocida como picudo de la guayaba afectó cultivos en Oiba (Santander) y por no tener controles ni información precisa, la plaga se extendió por varios municipios afectando la producción de guayaba y consecuentemente la de bocadillo. Desde el 2004 el ICA aumentó su presencia y cooperación en los cultivos de guayaba implementando planes de mitigación, entre ellos el control biológico y químico de plagas (ICA, 2014).

La guayaba es solo un ejemplo de cultivo que puede ser afectado por las plagas, según datos de la asociación nacional de café de Guatemala, (Anacafé), se estima que las pérdidas de café causadas solamente por la broca a nivel mundial alcanzaron los quinientos millones de dólares en el 2010. En Colombia, en abril de 2020, la federación nacional de cafeteros emitió una alerta temprana debido al contagio del 6.2% en cultivos,

cifra que ha venido en aumento y afecta tanto la calidad del grano como los niveles de producción.

Los cultivos de café son de gran importancia para la economía en Colombia. Durante el 2021 se exportaron 12.6 millones de sacos, y en 2020 13.9 millones de sacos, cada saco de 60 kilogramos, según la federación nacional de cafeteros de Colombia[1], mientras los precios en el 2022 han variado desde los 150 dólares a 231.45 dólares el saco, teniendo un máximo de 260.45 dólares en febrero ese año[2], lo que implica que el precio de la carga (125 kilogramos) varía de $1.870.000 a $2.670.000, por lo que se hace necesario poder desarrollar una herramienta que identifique las plagas que pueden afectar los cultivos, y poder determinar las prácticas adecuadas para el control de plagas a tiempo y evitar grandes pérdidas.

Es necesario entonces, ver las plagas como un problema que afecta una determinada región y a sus pobladores, porque éstas no obedecen a límites de terrenos o cultivos. Una solución está en las técnicas que les permiten a las computadoras aprender e imitar el comportamiento humano, conocidas como técnicas de inteligencia artificial, dentro de las cuales están los algoritmos de aprendizaje automático y profundo, capaces de enseñarle a la máquina a detectar plagas a partir de diferentes tipos de imágenes

[1] Fuente: https://federaciondecafeteros.org/wp/listado-noticias/produccion-de-cafe-de-colombia-cierra-2021-en-126-millones-de-sacos/

[2] Fuente: https://www.eleconomista.net/economia/Precio-del-cafe-subio-un-53--a-mayo-de-2022-20220620-0001.html

mediante el entrenamiento de redes neuronales que están formadas por interconexiones de unidades de procesamiento, también conocidas como neuronas.

Estas neuronas están interconectadas y se dividen en 3 capas: una capa de entrada una capa oculta y una capa de salida. La capa de entrada también es la encargada de recibir los datos o señales, la capa de salida es la encargada de proporcionar el resultado de las operaciones que realiza la red neuronal estas dos capas están conectadas al ambiente, ya que es necesario poder analizar los datos del entorno; la capa oculta tiene ajustes internos para poder responder al entrenamiento que se le ha hecho previamente (Vorobioff, Cerrotta, Eneas Morel, & Amadio, 20222).

Por medio de la inteligencia artificial, aplicando redes neuronales sería posible hacer una predicción de qué afectación está teniendo el cultivo de café. Para esto es necesario captar imágenes y hacer un procesamiento previo de estas, para encontrar la forma más eficiente en donde se pueda determinar el tipo de plaga que está afectando el cultivo y poder determinar cuál sería el control más efectivo para combatirla. Las imágenes obtenidas, de las hojas de las plantas, serán usadas para alimentar una capa de entrada en una red neuronal que determinará el tipo de plaga. Será necesario utilizar una red neuronal clasificadora y realizar un algoritmo que le permita aprender a esta red neuronal cómo diferenciar los diferentes elementos presentes en las imágenes los cuales podrían ser hojas pequeñas piedras tallos insectos u hongos.

Al poder realizar el entrenamiento de una red neuronal, que permita la identificación de plagas, y poder realizar por medio de un dispositivo móvil la adquisición de datos, servirá para una adecuada identificación de las plagas y como base para un sistema automático de cultivo de café, que permita disminuir los costos de producción y disminuir la cantidad de sustancias residuales en los granos de café.

1.3 Pregunta

¿Cómo mejorar la identificación de plagas como Cercospora, Phoma, Roya, minador de hojas y Arañita roja en cultivos de café por medio de imágenes?

1.4 Objetivos

1.4.1 Objetivo principal

- Implementar un aplicativo basado en visión artificial para la identificación de Cercospora, Phoma, Roya, minador de hojas y Arañita roja, en el café a partir del análisis de las hojas de la planta.

1.4.2 Objetivos secundarios

- Clasificar por medio de una red neuronal imágenes que identifiquen las plagas: Cercospora, Phoma, Roya, minador de hojas y Arañita roja en los cultivos de café.

- Implementar una interfaz gráfica que permita la adquisición y visualización de resultados.

1.5 Antecedentes

A continuación, se referencian trabajos que se han realizado en la línea de investigación relacionados con algoritmos de clasificación de imágenes en el sector agrícola, en especial la identificación de insectos y plagas.

En la universidad pedagógica Nacional, (Muñoz, 2019) utiliza técnicas de procesamiento de imagen para la clasificación de la calidad de rosas. Tal clasificación se hace considerando los requerimientos del mercado como lo es: enfermedad vellosa, malformaciones de tallo y deformidades en los botones. Para ello plantea dos etapas de trabajo, una para el preprocesamiento de imágenes y la segunda para el análisis de la imagen con un algoritmo de inteligencia artificial, para lo cual se utilizó la herramienta de Matlab: *classification learner,* la cual implementa un algoritmo de aprendizaje supervisado para la detección de la calidad de la rosa. En cuanto a la presentación de resultados: el porcentaje de predicción del tamaño del botón y si es tipo de exportación se da mediante una interfaz gráfica desarrollada en GUI de Matlab. Como resultado de la implementación de la inteligencia artificial, se obtiene una precisión del 22% en la predicción de la calidad general de la flor (Muñoz, 2019).

En cuanto a la detección de algunos insectos en cultivos de caña de azúcar, Thenmozhi y Reddy desarrollan un algoritmo basado en el análisis de imágenes. Este

análisis se hace por medio de Matlab para poder extraer las características de las imágenes que son tomadas en RGB. Éstas se convierten a escala de grises para posteriormente aplicar los métodos de detección de bordes Canny, Sobel, Gaussian HPF y Prewit. Por medio del análisis de figuras básicas como círculos y triángulos, se calculan en área y el perímetro del insecto, y por medio de la relación área y perímetro se hace una aproximación a la detección de éstos (Reddy, 2017).

Por medio de imágenes aéreas tomadas por drones a una distancia de 10, 15 y 20 metros como lo propone (Olivera, 2019), se han identificado las afectaciones en cultivos de café por medio de redes neuronales, para lo cual realizaron una base de datos con las imágenes que identificaron y segmentaron para tener los sets de entrenamiento y de prueba. Estas imágenes tuvieron que ser clasificadas con ayuda de un experto, logrando entrenar un modelo para identificar afectaciones en cultivos extensos (Oliveira, 2019).

Los modelos de las redes neuronales pueden ser creados con las librerías TensorFlow y Keras, los cuales están soportados para diferentes lenguajes de programación. El modelo creado con estas bibliotecas puede ser exportado y usado en otras plataformas como los soportados por Android o aplicaciones de escritorio sin necesidad de realizar una nueva compilación del modelo neuronal. En el caso de (Valencia, 2022), se realizó un modelo neuronal que puede detectar las expresiones faciales de las personas, modelo que fue exportado a una aplicación de escritorio y a una aplicación soportada en Android (Valencia, 2022).

Los modelos neuronales que son creados por medio de la librería de TensorFlow, pueden ser exportados en una versión más liviana y usados en aplicaciones para Android o iOS, como lo desarrolla Casa Robles en su trabajo de maestría donde crea un modelo para la clasificación de imágenes basado en una red neuronal usando TensorFlow Lite con Python, y probando diferentes tipos y configuraciones de las redes neuronales. Tales modelos se evaluaron y seleccionaron para ser implementados en una aplicación para Android, desarrollada en Android Studio y Bazel (Casa Robles, 2020).

Para identificar diferentes tipos de estrés en las hojas en el café arábico, en (Esgario, 2020) crearon una base de datos de 2147 imágenes, las cuales se tomaron con diferentes tipos de celulares y fueron clasificadas dependiendo de las enfermedades en: minador de hojas, roya, cercospora, phoma, y hojas sanas por un experto. Tales imágenes fueron usadas para la creación de un modelo de red neuronal multitarea basado en las arquitecturas AlexNet, GoogLeNet, y ResNet50, llegando a tener resultados cercanos al 80% de exactitud, presentando sin embargo, posibles errores al momento de realizar las predicciones, puesto que las enfermedades sobre las cuales se hizo la predicción no fueron lo suficientemente amplias (Esgario, 2020).

Capítulo 2

2.1. Marco Teórico o de Referencia.

2.1.1 Plagas Del Café.

El cafeto es una planta que se desarrolla a la sombra y en ambientes tropicales donde la humedad debe ser superior al 70%, la temperatura media debe estar entre los 17°C y 23°C, condiciones que no solo son benéficas para las plantas, sino que también para las plagas que viven de varios cultivos y entre ellos el cafeto. Entre los diferentes tipos de enfermedades que puede tener el cultivo se pueden tener las producidas por insectos y por hongos, las cuales se manifiestan de diferentes formas, entre ellas la afectación que se produce en las hojas, en el tallo o en los frutos.

2.1.1.2 Minador de hojas

El minador de hojas es una mariposa que se ha adaptado a las condiciones ambientales en Colombia, fue vista por primera vez en 1842 en las islas Guadalupe y Martinica. Es un insecto originario de las islas Reunión al este de África. El cambio climático ha permitido que esta plaga se adapte de cultivos situados a los 1300 metros sobre el nivel del mar a los cultivos ubicados entre los 1500 y 1700 metros sobre el nivel del mar, en algunos municipios de Caldas, Cauca, Tolima y Valle del cauca (Constantino L. M., 2013).

En su estado adulto es una mariposa de entre 2 y 3 mm de longitud y 4.2 a 4.3 mm de envergadura, las alas son de color negro con un punto gris y cuerpo blanco, (Enriquez,

1975). Sus hábitos son nocturnos, aunque vuelan en días nublados (Constantino L. M., 2013).

La afectación a la planta del café se da cuando la larva se alimenta de la hoja, la cual puede consumir entre 1 a 2 mm^2 por día (Constantino L. M., 2013). Debido a la acumulación de excrementos se presentan zonas de afectación donde la zona afectada de la hoja va cambiando la tonalidad verde claro a un café marrón claro o negruzco (Enriquez, 1975).

Figura 1

(a) minador de hoja (b) afectación del minador (c) café afectado por minador de hojas

Nota De Constantino, L. M. (2013). Minador de las hojas del cafeto: Una plaga potencial por efectos del cambio climático. Manizales: Centro Nacional de Investigaciones de Café (Cenicafé).

2.1.1.3 Roya

La roya es un tipo de hongo que se propaga por esporas que son transportadas por el aire, animales, ropa o agua, donde llega a la planta y empieza su ciclo de vida para lo cual requiere una temperatura entre 15°C y 28°C (Hernández-Martínez, 2016), poca o nula intensidad lumínica y una capa de agua. Una vez llega el hongo a la hoja, de 21 a 24 días, este desarrolla estructuras que se ponen en contacto con las células de la planta

para alimentarse de los nutrientes de ésta produciendo manchas amarillentas, generando pérdidas en la producción de la planta de café y disminución del tamaño de los granos (Rivillas, 2011).

Figura 2.
Afectación de Roya

Nota. De Rivillas, C. A., Leguizamón, J. E., & Gil, L. F. (1999). Recomendaciones para el manejo de la roya del cafeto en Colombia.

2.1.1.4 Phoma spp

Phoma spp es un hongo que causa la enfermedad que se conoce como muerte descendiente. Suele afectar a cultivos que se encuentran a alturas superiores de los 1.600 metros sobre el nivel del mar con condiciones climáticas de bajas temperaturas, vientos fríos, y baja exposición a brillo solar, causando la muerte de brotes y hojas, daño que va descendiendo en la planta hasta llegar a las zonas de crecimiento, limitando la capacidad productiva de la planta y dificultando que se generen nuevos brotes (Marín-Ramírez, 2019). Una vez el hongo infecta la planta, los primeros síntomas aparecen a los 4 días donde se ven manchas negras en las hojas; a los 10 días las manchas se van uniendo y finalmente mueren completamente a los 72 días, donde comienzan a emitir esporas que son transportadas a otras plantas, mientras tanto la enfermedad sigue

atacando a la planta descendiendo por los tallos los tallos y eliminando nuevos brotes (Gil-Vallejo & LeguizamÛn-Caycedo, 2000).

Figura 3
Afectación de Phoma

Nota. De Cenicafé 2000, la muerte descendente del cafeto (phoma spp.).

2.1.1.5 La Broca del café

La Broca del fruto del café, *Hypothenemus hampei*, es la plaga de mayor impacto en Colombia y fue introducida en 1988. Es una plaga que ataca al fruto del café perforándolo y provocando su caída prematura. Esta plaga fue introducida desde Brasil, a donde llegó en 1913 (BENAVIDES M, 2013). El café, en su hábitat natural, es una planta que crece a la sombra de árboles grandes, por lo que se cree que la broca es un insecto se ha adaptado a estas condiciones de sombrío. Este insecto se traslada entre plantas por medio del vuelo de las hembras, las cuales pueden volar hasta tres horas en forma intermitente; los machos solo tienen función reproductiva y a pesar de tener alas no vuelan. La hembra adulta es fecundada por el macho y empieza el proceso de búsqueda de nuevos frutos, los cuales identifica por medio del olor y el color. Éstas ingresan a los

frutos que se encuentren verdes a través de la corona o de la cicatriz floral, llegando así a la nuez para formar galerías donde depositan entre 10 y 30 huevos (Constantino L. M., 2011), permaneciendo allí hasta su muerte; una vez las larvas comienzan su alimentación comen el fruto hasta que salen de éste. Si el fruto no es cosechado, se cae al piso y continua el ciclo.

Figura 4.
Café afectado por la broca

La morfología de estos insectos en estado adulto es de color negro brillante, de forma cilíndrica; suelen tener un tamaño de 1.55 mm x 0.82 mm para el macho y 1.12 mm x 0.53 mm para las hembras; los huevos son de color blanco lisos y tienen una elíptica ovalada de 0.7 mm x 0.3 mm (Constantino L. M., 2011).

El control por medio de plaguicidas ha sido poco eficiente ya que el ciclo de vida de este insecto es continuo y una vez ingresa al fruto no es posible la aplicación del insecticida, además que la cantidad de insecticida que se requiere para hacer el control pondría en riesgo la salud humana debido a la forma en la que se cultiva el café y la

relación con las personas. Como en Colombia se presenta floración y brotes de fruto a lo largo del año, los cultivos son susceptibles a ataques constantes. Entre las estrategias que se plantean para el control de esta plaga se encuentran la introducción del hongo Beauveria bassiaba o avispas de origen africano (Bustillo Pardey, 2007).

2.1.1.6 Pasador de las ramas de café

El pasador de ramas de café, *Xylosandrus morigerus Blandford,* es considerado como una plaga limitante en la producción de café en variedades diferentes al café arábico, a pesar de ser un insecto que consume un hongo en específico, tiene una relación simbiótica con el hongo. Su origen es Indonesio, pero por los intercambios comerciales ha llegado a afectar los cultivos mundiales, en Colombia se identificó en 1957. Este insecto perfora las ramas del café donde realiza los nidos en los que se reproducen y nacen las crías; luego del desarrollo de las larvas, las hembras salen a crear nuevos nidos y con ello llevan el hongo, el cual crece en las paredes del nido (Jaramillo, Benavides machado, & Constantino, 2015).

En cuanto a su anatomía, el pasador de ramas es un insecto de 1.7mm de largo por 0.8mm de ancho en el caso de las hembras, mientras que el tamaño promedio del macho es de 1.0 mm de largo por 0.5mm de ancho (Jaramillo, Benavides machado, & Constantino, 2015). Las hembras cuentan con alas funcionales que les permiten hacer vuelos para llegar a nuevas ramas y continuar con el ciclo de vida, el cual está entre los 20 y 40 días.

Cuando este insecto llega a una planta de café, hace una perforación en las ramas donde forma una galería, impidiendo el flujo de la sabia hacia los frutos y hojas, provocando una muerte parcial de las ramas y hojas. Este insecto en ocasiones introduce organismos que afectan al café, o las galerías desocupadas sirven de nido para hormigas, y hay casos donde la rama se recupera, mientras que la mayoría de veces, la rama pierde capacidad de producción y se quiebra con facilidad (Jaramillo, Benavides machado, & Constantino, 2015).

Figura 5

Pasador de ramas. (a) Orificio hecho por pasador de ramas, (b) nido de ninfas de pasador de ramas, (c) afectación a ramas

2.1.1.7 Cochinillas harinosas

Las cochinillas harinosas o palomillas, son insectos que se alimentan de material vegetal. Es una plaga endémica que ha afectado otros cultivos y plantas silvestres (Gil-Palacio, 2021) y bajo las condiciones adecuadas afecta también al café desde las raíces de los árboles, limitando la producción y provocando afectación en los cultivos. Esta plaga suele ser confundida con los pasadores por la afectación, pero se diferencia en que ésta se alimenta de la savia de las plantas provocando su muerte a partir de las raíces (BENAVIDES M, 2013).

La afectación a los cultivos de café por parte de estos insectos se registró desde 1930 en Antioquia, causando afectación en los cultivos hasta el día de hoy. Esta plaga suele atacar las plantaciones menores de dos años. Desde 1930 cuando se registró la variedad *Puto antioquiensis,* se han registrados diferentes variedades de plagas llegando a afectar a la mayoría de los departamentos cafeteros del país (Gil-Palacio, 2021). En la Tabla 1, se muestran las principales especies y su afectación en los cultivos en los departamentos de Antioquia, Caldas, Risaralda, Quindío, Valle del Cauda Tolima y Cauca. Los datos suministrados por Gil-Palacio fueron muestreados a partir del estudio de campo en fincas cafeteras, identificando la afectación en las plantas y la presencia de insectos en éstas. Entre los principales insectos que afectan los cultivos se encuentran: *Dysmicoccus spp, Geococcus caffeae, Puto Barberi, Rhizoecus spp, Ripersiella andensis.*

Tabla 1
Diagnóstico de las conchillas

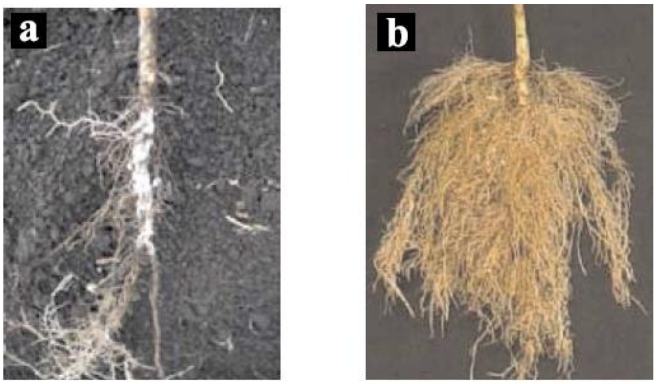
		Departamento							
		Antioquia	Norte de Santander	Caldas	Risaralda	Quindío	Valle del Cauca	Tolima	Cauca
Fincas con presencia		77.2%	43%	74.2%	84.2%	87.4%	56.6%	71.7%	75.3%
Número de especies		14	12	16	15	16	14	15	16
Especie	*Dysmicoccus* spp.	22.8%	24.4%	41.2%	28.4%	48.4%	17.6%	34.4%	27.3%
	Especies que enquistan	28.3%	4.7%	38.1%	22.1%	40%	23.5%	26.3%	32.5%
	Geococcus caffeae	38%	1.2%	51.5%	58.9%	45.3%	8.2%	32.3%	53.2%
	Mixorthezia minima		4.7%						
	Neochavesia caldasiae	6.5%		1%	6.3%		1.2%		
	Phenacoccus solani						1.2%	4%	
	Planococcus spp.			2.1%				14.1%	
	Pseudococcus spp.	12%		10.3%	13.7%	31.6%	15.3%		16.9%
	Pseudorhizoecus bari		1.2%						
	Puto barberi	37%	30.2%	26.8%	43.2%	85.3%	44.7%	23.2%	41.6%
	Rhizoecus spp.	21.7%	2.3%	26.8%	45.3%	28.4%	3.5%	13.1%	26%
	Ripersiella andensis	3.3%		8.2%	12.6%	12.6%	2.4%	5.1%	
	Toumeyella coffeae		14%				3.5%		1.3%

Estos insectos que se alimentan de la savia de las plantas, tienen un tamaño que varía desde los 1.5 mm hasta 5.0 mm de largo y de 0.5 mm hasta 2.5 mm de ancho. Su forma suele ser ovalada y aplanada; su cuerpo tiene una capa protectora blanca y cerosa constituida por ácidos grasos. El ciclo de vida puede variar entre los 78 y 54 días, con temperatura entre 21.8°C y 28.9°C y humedad relativa entre 75% y 80%; en estas condiciones se pueden llegar a presentar hasta 15 generaciones en un año (Villegas C. Z., 2010) (Villegas C. P., 2015).

La forma de atacar las plantas consiste en parasitar la base del tronco y las raíces. La profundidad de la afectación varía dependiendo del tipo de suelo desde los 10 cm hasta los 60 cm. Al cubrir las raíces de la planta, se limita la absorción de nutrientes, lo que va causando un debilitamiento, llegando a causar la muerte (Villegas C. Z., 2010). Los síntomas que se evidencian son: “debilitamiento general, retraso en el desarrollo, desprendimiento de la corteza, tallos delgados, entrenudos cortados, caída prematura de los frutos” (Villegas C. Z., 2010). Por la forma en la se evidencia el ataque se puede confundir con otro tipo de afectaciones, como hongos, deficiencias nutricionales o afectaciones ambientales.

Figura 6

(a) Raíz de cinco meses con presencia de cochinillas harinosas. (b) Sin presencia de cochinillas harinosas.



Nota. De: Villegas, C., Zabala, G. A., Ramos, A. A., & Benavides, P. (2010). Identificación y hábitos de cochinillas harinosas asociadas a raíces del café en

2.1.1.8 Cercospora

Entre los hongos que pueden afectar a los cultivos de café se encuentra el Cercospora, el cual causa la enfermedad que se conoce como mancha de hierro. Este

hongo ataca el fruto y las hojas del café, dejando manchas generalmente circulares de color marrón con un tamaño que puede variar entre un milímetro y 3 milímetros, y es más dañino en las temporadas secas (Rengifo, 202).

Figura 7
Daño ocasionado por Cercospora.

Nota. De: Rengifo, H. G., Leguizamon, J. E., & Riaño, N. M. (2002). Algunos aspectos biológicos de Cercospora coffeicola.

2.1.1.9 Araña roja

Este insecto es un ácaro de aspecto similar a una araña roja que no es visible a simple vista. Tanto los adultos como las ninfas se alimentan de la savia de las hojas, lo que ocasiona despigmentación de la hoja ya que, al consumir el contenido celular de ésta, se pierde el brillo y se tornan bronceadas. Esta plaga es endémica y se manifiesta en focos pequeños y está presente tanto en la parte superior como inferior de la hoja, en donde tejen una telaraña que sirve de nido para la incubación de huevos y que además limita el ingreso o salida de partículas. Si no se realiza el control necesario rápidamente, afecta la plantación completa (GIL, 2013).

Figura 8

Daño ocasionado por la Araña roja.

Nota: imagen tomada de: GIL, Z., Constantino, L. M., Martínez, H., & Benavides, P. (2013). Aprenda a manejar la arañita roja del café. Centro Nacional de Investigaciones de Café (Cenicafé).

En la Tabla 2 se presenta un resumen de la afectación que tienen as plagas en la planta de café, para poder determinar que las plagas que se tendrán en cuenta para el desarrollo de la red neuronal, las cuales son: Minador de hoja, Roya, Phoma, Cercospora y Araña roja.

Tabla 2
Cuadro comparativo afectaciones del café, broca, pasador de ramas y cochinillas harinosas

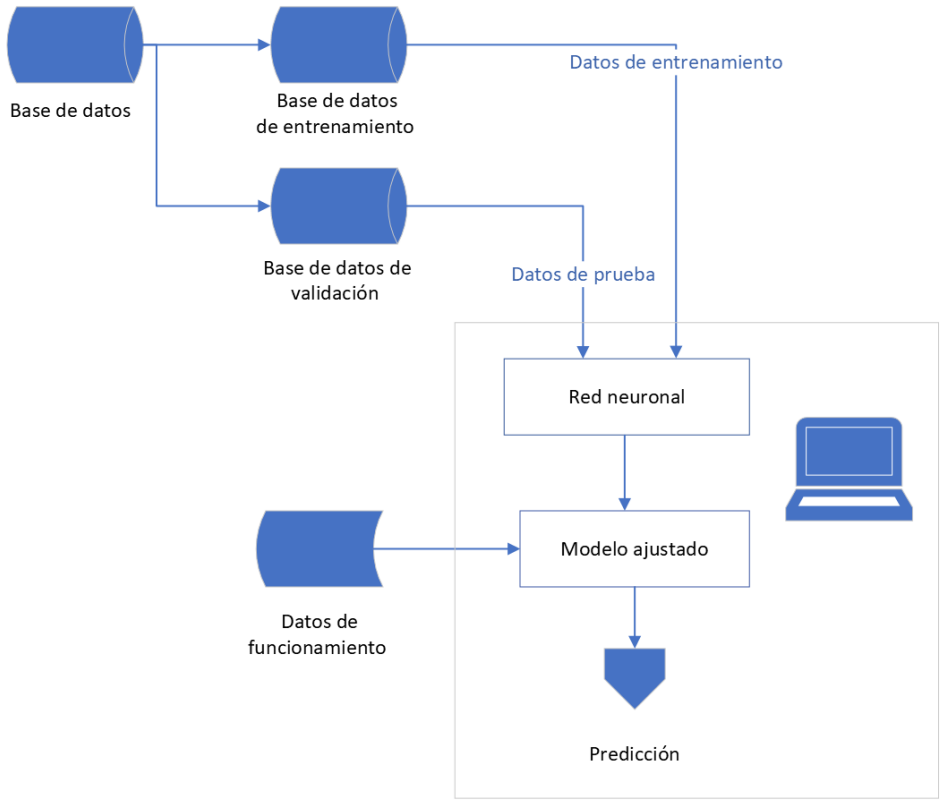
Plaga	Fuente de alimentación	Principal afectación	Afectación a hojas	Afectación al fruto	Afectación al tallo
Broca (insecto)	Cereza	Daño en fruto	Nulo	Perforación en cereza	Nulo
pasador de ramas (insecto)	Savia de las plantas	Perforación en ramas	Amarillamiento y perdida de hojas	Perdida del fruto	Daños solos visibles en tallos afectados
cochinillas harinosas (insecto)	Savia de las plantas	Perdida de sistemas de raíces	Amarillamiento y perdida de hojas	Perdida del fruto	Debilitamiento
Minador de hoja (insecto)	Hoja	Zonas cafés en hojas	Zonas cafés	Disminución de producción	Nulo
Roya (hongo)	Savia de las plantas	Zonas amarillas en hojas	Zonas amarillas	Disminución de producción	Nulo
Phoma (hongo)	Hoja	Zonas negras en hojas	Zonas negras	Zonas negras	Tallos muertos
Cercospora (hongo)	Hoja	Zonas marrones en las hojas	Zonas marrones	Zonas marrones	Nulo
Araña roja (insecto)	Savia de las plantas	Daño en hojas	Perdida de brillo y bronceado	Disminución de frutos	Nulo

2.1.2 Aprendizaje de máquina

El aprendizaje de máquina se basa en una serie de algoritmos que, apoyados en análisis estadísticos, son capaces de clasificar o predecir a que tipo pertenecen los elementos de un conjunto de datos llamados datos de prueba, a partir de un conjunto de datos comúnmente denominados datos de entrenamiento. A diferencia de las estructuras convencionales de programación que se basan en el uso de estructuras condicionales o de bucle, con los que se hacen algoritmos basados en la toma de decisiones establecidas por un programador que tiene bien definido las condiciones en el flujo de información, el aprendizaje de máquina se basa en mejorar los mecanismos de decisión basándose en la información cambiante. (Müller & Guido, 2018)

Adicionalmente, un algoritmo de aprendizaje de máquina trata de usar un modelo ya definido y realizar ajustes en sus parámetros internos hasta lograr la predicción adecuada, proceso que se conoce como aprendizaje. Entre los algoritmos de aprendizaje se encuentran: los sistemas de aprendizaje supervisados, los cuales necesitan una base de datos que contiene datos de entrenamiento que deben estar etiquetados o categorizados para que el modelo pueda establecer adecuadamente los patrones que se quieren identificar. Además, se requiere de una segunda base donde se tienen datos de prueba, de los cuales también se conoce el resultado y son usados para validar el funcionamiento del modelo (por lo general se toma una única base de datos y esta se divide en dos, donde un porcentaje cercano al 20% se establece como datos de validación). Una vez que el modelo tiene un grado de predicción deseado, se puede poner en funcionamiento. Para poder establecer mejor el flujo de información entre el modelo y las bases de datos, se presenta la Figura 9, en la cual se muestra que las bases de datos sirven de entrada a una red neuronal, que una vez ajustada, se usa para realizar las predicciones necesarias.

Figura 9
Modelo de funcionamiento algoritmo de aprendizaje supervisado.



El otro modelo de entrenamiento es el de aprendizaje no supervisado, el cual consiste en un proceso de ajuste a los valores internos de un modelo para llegar a un valor de predicción adecuado. A diferencia del aprendizaje supervisado, los datos no están categorizados ni etiquetados, sino que éstos se encuentran agrupados y el modelo se encarga de identificar los patrones que tengan en común los grupos de datos.

2.1.3 Redes Neuronales Artificiales

Las redes neuronales artificiales se asemejan al funcionamiento del cerebro, ya que son unidades de memoria básicas en las cuales se almacena un tipo de información y están interconectadas con otras neuronas para poder hacer un procesamiento por medio

del flujo de señales entre ellas. Estas señales funcionan como un sistema de conexiones, las cuales llevan información externa a las neuronas y también entre las neuronas que están en las diferentes capas. Los modelos neuronales pueden ser de una sola neurona o la unión de varias neuronas en varias capas (Bertona, 2005).

Cuando un modelo contiene una única neurona, esta se comporta como la ecuación de una recta y contiene un único canal de entrada y de salida. Éstas tienen un sesgo y un peso que ayudan a calcular los valores necesarios de la red. Si se trata de la unión de varias neuronas, estas se deben agrupar en capas y se debe tener una capa de entrada donde se cuenta con una neurona por cada dato de entrada. A la salida se debe tener una neurona por cada salida que se requiera. Entre estas dos capas hay una capa adicional que se conoce como capa oculta, la cual es la encargada de hacer el procesamiento matemático. Esta red de neuronas artificiales es capaz de combinarse de diferentes formas a partir de señales y procesos de entrenamiento para obtener comportamientos que satisfagan los datos de entrada y salida que se usan para este entrenamiento (Bertona, 2005).

Los sistemas multicapas son sistemas complejos donde se conectan *n* cantidad de neuronas, todas interconectadas con las neuronas de la capa anterior y la siguiente, lo cual permite mejorar el cálculo y tener aproximaciones más exactas. Los entrenamientos de la red neuronal son hechos a partir de estímulos de entrada – salida. Entre los procesos de aprendizaje está el cálculo de perceptrones. El proceso de aprendizaje consta del cálculo de pesos y sesgos en la red a partir de datos de entrada y salida, y al

finalizar la suma de todos los valores se obtiene un modelo (Vorobioff, Cerrotta, Eneas Morel, & Amadio, 20222).

2.1.3.1 Modelos lineales

Entre los modelos de redes neuronales se encuentran los modelos lineales, los cuales tratan de dar la aproximación a un conjunto de datos por medio de la sumatoria de varias rectas. Se usan para la predicción de datos bidimensionales, como por ejemplo los datos de una curva o datos de un sensor. Para ello, se toma un modelo que se describa a partir de ecuaciones lineales de la forma:

$$Y = \sum_{i=0}^{n} W_n X_x + B$$

Donde $W_n$es la pendiente de cada una de las rectas, X_x los valores de la variable independiente, y B la suma de las intercepciones con el eje y cuando x=0; ecuación que se puede simplificar a la ecuación de la recta con n=1.

Otro tipo de ajuste que se suele emplear en los modelos neuronales es la activación de neuronas, donde se impone un límite o una determinada función matemática para que la información pueda seguir a otra neurona, y poder descartar neuronas que no sean relevantes para el modelo.

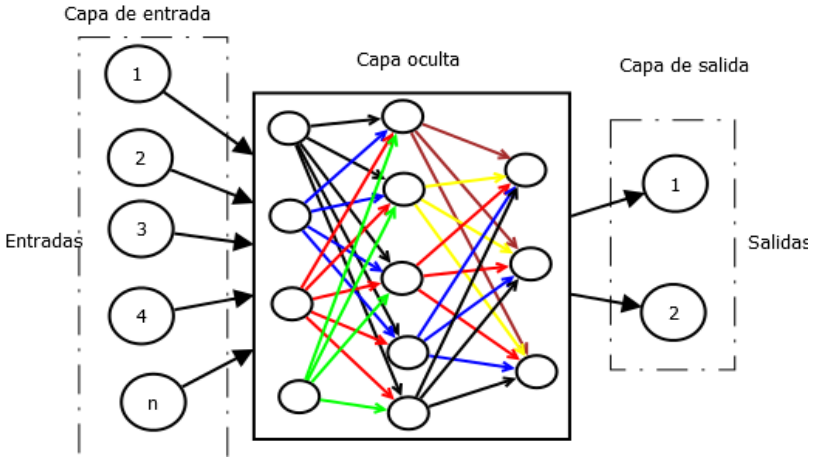
2.1.3.2 Redes neuronales densas

Las redes neuronales densas son uniones de varias capas, las cuales están conformadas por múltiples neuronas que se interconectan entre capas. La conexión se da entre una neurona y todas las neuronas de la capa siguiente, teniendo como inicio la capa de entrada con un tamaño igual a la cantidad de datos que va a tener el modelo. Por ejemplo, para una imagen se tomaría la cantidad de píxeles y una neurona para cada uno en la capa de entrada, y para la capa de salida se tendría una neurona por cada categoría o dato de salida que se requiera. En las capas intermedias la cantidad de neuronas se establecen de acuerdo al tamaño que se quiera tener del modelo, entre más capas se tengan, el modelo será más complejo y tomará más tiempo el entrenamiento, pero no se asegura que la predicción sea mejor (Torres, 2020).

Una representación gráfica de la red neuronal que se usaría para el entrenamiento de máquinas es la mostrada en la Figura 10, donde se define claramente la capa de entrada, la capa de salida y las capas ocultas que es donde están las capas intermedias:

Figura 10

Modelo de red neuronal densa.



Las neuronas de las capas ocultas pueden ser o no activadas. Con este proceso lo que se hace es diferenciar las neuronas que pueden tener un valor significativo en el modelo de las que simplemente aumentan el error. Al finalizar, se formará un camino donde la información fluirá activando neurona-neurona entre capas logrando así un resultado de predicción.

2.1.3.3 Redes neuronales convolucionales

Las redes neuronales convolucionales o CNN por sus siglas en inglés (Convolutional Neural Networks) son usadas desde los años 1990 y se constituyen en una aproximación a la forma en que las personas identifican objetos, tratando de imitar el proceso en el cual se identifican características básicas de una imagen como lo son: ejes, bordes, y los modelos de los objetos. En la formación de este modelo neuronal convolucional se

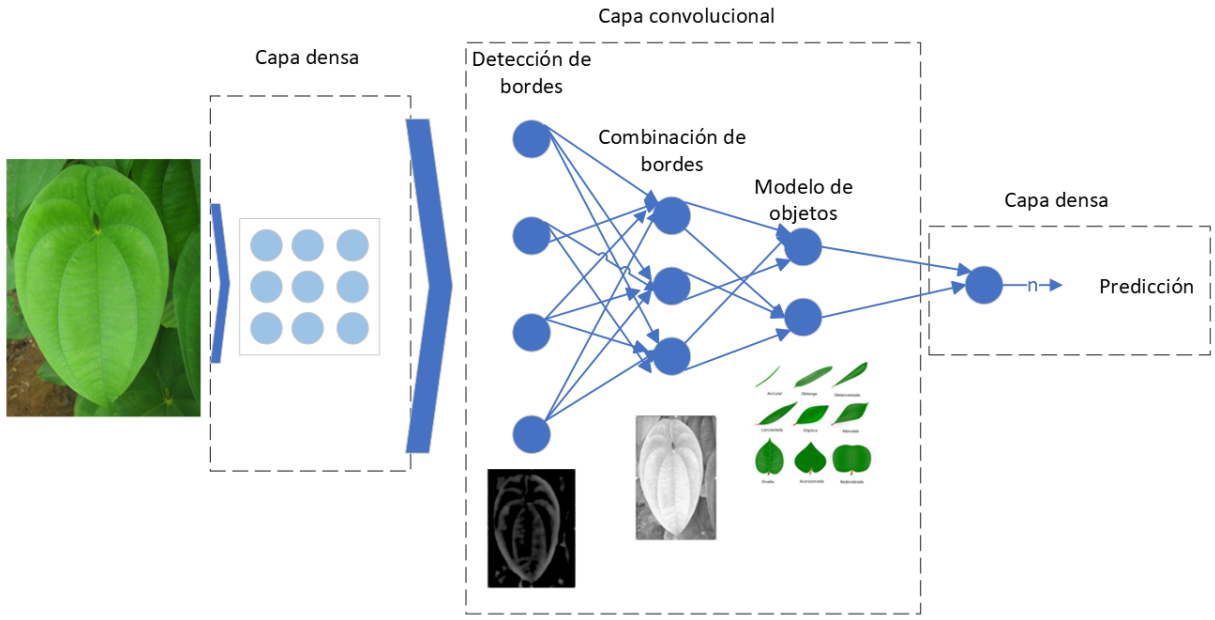
implementa una capa convolucional que aprende patrones en las imágenes y conectada a capas densas (Torres, 2020).

En estas redes, al tener una estructura más compleja que las redes densas, el tiempo de entrenamiento es mayor pero se logra identificar mejor las características de las imágenes, ya que pueden identificar patrones globales y características que pueden estar ubicadas en cualquier parte de la imagen, mientras que una red densa únicamente las identifica de manera local, es decir, debe estar siempre en la misma posición para poderla identificar.

Una representación gráfica de este tipo de redes se muestra en la Figura 11, donde la red neuronal convolucional está construida con dos capas densas, una de entrada donde se tiene una neurona por cada pixel de la imagen, una capa de salida donde se tiene una neurona por cada categoría del entrenamiento (n), y la capa oculta que está conformada por capas convolucionales, las cuales con la forma en la que están construidas pueden aplicar filtros para la detección de características, como detección de bordes, combinación de bordes y modelado de objetos. Combinando estas tres capas se puede detectar patrones con las imágenes de la base de datos.

Figura 11

Modelo de red neuronal convolucional.



2.1.4 overfittting,

El sobreajuste o overfiting, es el resultado de hacer entrenamientos consecutivos con las mismas imágenes, que da como resultado que la red neuronal memorice los datos de ingreso en vez de encontrar una regla general en los datos, generando un ruido en la predicción la cual se nota a medida que transcurren las épocas como un descenso en la predicción de los datos tanto en el entrenamiento como en la validación; para hacer una adecuara corrección de este ruido existen varias técnicas como validación cruzada, validación cruzada o métodos de regulación, cuando se trabaja con imágenes una forma de realizar la corrección es tomar las imágenes de entrenamiento y realizar transformaciones que resulten reales, como girar las imágenes, y reflejar las imágenes, con ello se trata que la red a busque patrones globales en las imágenes. (Dietterich, 1995)

2.2 Marco de implementación

El aprendizaje de máquina se puede implementar diferentes lenguajes. Para el desarrollo de este trabajo se utilizó el lenguaje Python con sus bibliotecas TensorFlow y Keras, ya que estas permiten una fácil conexión con los dispositivos móviles por medio de la librería TensorFlow Lite, cuyos modelos se pueden importar a un proyecto java por medio de Android Studio.

2.2.1 Python

Es un lenguaje de programación multiparadigma, creado por Guido van Rossum en Stichting Mathematisch Centrum a inicios de la década de 1990. Es un lenguaje de código abierto disponible bajo licencia publica general GNU-GPL (Python Software Foundation, 2022). Para realizar la interpretación de este lenguaje es necesario usar una distribución de Python y un entorno de desarrollo como Visual Studio, Spyder, o Jupyter, entre otros.

2.2.2 NumPy

Esta biblioteca desarrollada para Python que permite utilizar matrices multidimensionales, con la condición de que todos los elementes deben ser del mismo tipo, con el fin de poder realizar operaciones matemáticas y optimizar la ejecución del código (NumPy, 2022). Para realizar la importación de la biblioteca se utiliza la sentencia:

import *numpy* ***as*** *np*, (*np* puede variar).

2.2.3 TensorFlow

Es una librería que desarrolló Google Brain inicialmente con el nombre de DistBelief en el 2011. En el 2015 fue lanzado como código abierto TensorFlow, enfocada en el aprendizaje automático el cual puede operar en dispositivos móviles al igual que en sistemas distribuidos de dispositivos, como servidores o tarjetas GPU, para poder desarrollar sistemas flexibles ya que incluye varios modelos de redes neuronales profundas permitiendo aplicaciones en varios campos de investigación (Martín Abadi, 2015). Para realizar la importación de la biblioteca se utiliza la sentencia:

import *tensorflow* ***as*** *tf*, (*tf* puede variar).

2.2.4 TensorFlow Lite

La implementación de los modelos neuronales en dispositivos móviles, como tabletas, celular o microcontroladores es posible mediante las herramientas que presenta TensorFlow Lite, ya que convierte el modelo neuronal te TensorFlow en un modelo de ejecución ligera, ya que presenta cinco ventajas, las cuales son: (TensorFlow, 2022)

1. Mejora los tiempos de latencia al no tener comunicación con un servidor ya que el modelo se ejecuta directamente sobre la arquitectura del dispositivo.
2. No requiere una conexión permanente a internet.
3. Los datos no son exportados fuera del dispositivo.
4. El tamaño del archivo es menor.
5. Reduce la cantidad de energía para su ejecución.
6. Es compatible con diferentes plataformas, con dispositivos iOS, Android, ARM o Linux y lenguajes de programación como Java, C++, o, Python.

2.2.5 Keras

Keras es una interfaz de programación de aplicación (API) de aprendizaje profundo que permite la construcción de modelos neuronales. Fue escrita en Python para ser ejecutada mediante TensorFlow. Permite simplificar el proceso de creación del modelo neuronal al simplificar las sentencias necesarias, pero sin sacrificar capacidad de cálculo (Keras, 2022). Para realizar la importación de la biblioteca se utiliza la sentencia:

from tensorflow ***import*** *keras*

Capítulo 3

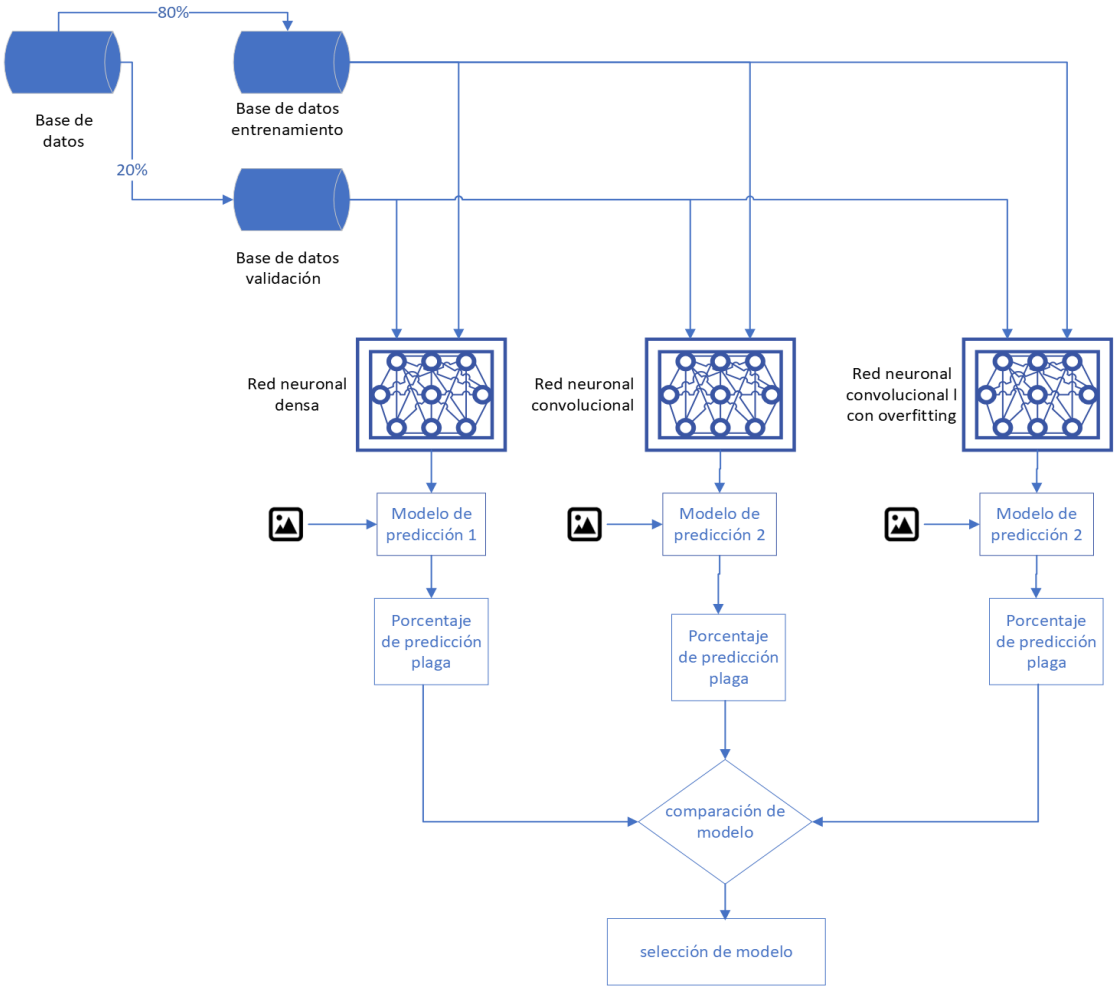
3.1 Metodología

La investigación consiste en entrenar un modelo neuronal que permita la identificación de las plagas: broca, pasador de ramas y cochinilla arenosa, a partir de bases de datos ya existentes. La base de datos está conformada por imágenes de hojas de café afectadas por plagas. Esta base de datos tiene imágenes en un ambiente controlado, esto es, la hoja de café con un fondo blanco. Las imágenes fueron obtenidas en campo por: J. G. M. Esgario, R. A. Krohling, J. A. Ventura. Esta base será introducida a modelos de redes neuronales: densa, neuronal convolucional y neuronal convolucional con overfittting, usando el lenguaje Python y las bibliotecas: Keras y TensorFlow. Una vez entrenado el modelo, se procede al análisis de las imágenes que contienen la morfología de la planta y que servirán de entrada a una red neuronal para poder comparar los resultados y determinar cuál de los modelos presenta mejores resultados. El modelo gráfico de entrenamiento es el mostrado en la Figura 12, donde la base de datos será divida en 2, un 80% se usa para en entrenamiento y el restante 20% para la validación

de los modelos. En los tres modelos, los de datos de validación y entrenamiento son los mismos.

Figura 12

Metodología de entrenamiento de la red neuronal



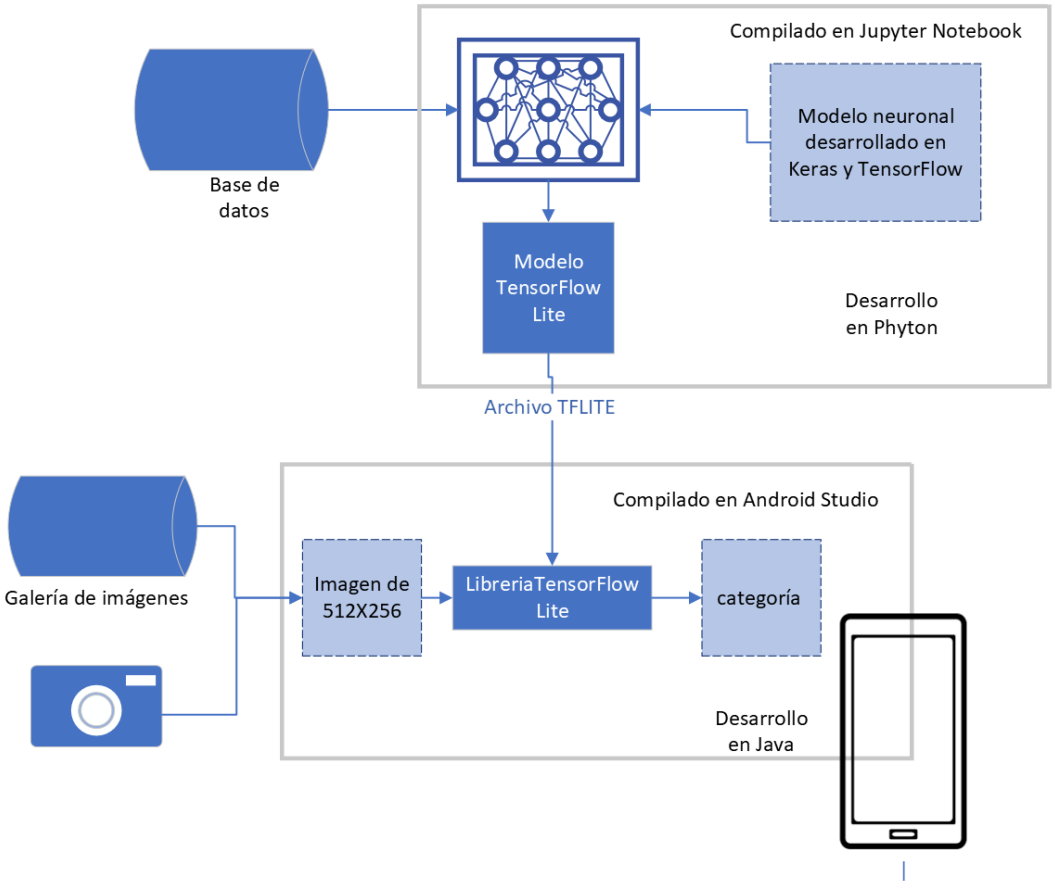
Una vez entrenado el modelo neuronal, es necesario exportar este modelo a un archivo de TensorFlow lite, el cual se puede ser importado a una aplicación soportada para Android por medio de Android Studio. La aplicación debe ser programada en

lenguaje java, importando las librerías correspondientes a TensorFlow y los permisos de usuarios para el acceso a multimedia, cámara y galería de imágenes.

El modelo es inicializado en Java y recibe como entrada una imagen, la cual debe ser escalada al tamaño que se tomó como referencia durante el entrenamiento de la red neuronal. Como salida, el modelo entrega la categoría que identifica con mayor precisión. En la Figura 13 podemos ver el flujo de información que tendría el entrenamiento y la aplicación, donde el desarrollo en Python da como resultado un archivo de TensorFlow Lite, que sirve como base en la aplicación de Java para la identificación de las imágenes.

Figura 13

Implementación del modelo neuronal una aplicación Android



Capítulo 4

4 Desarrollo y resultados

A continuación, se presentan los momentos en el desarrollo del proyecto, las actividades realizadas y los resultados que se obtuvieron en cada momento durante la realización del proyecto.

4.1 Momento 1: Base de datos

En una primera instancia se realizó la búsqueda de una base de datos que contuviera hojas que café con diferentes tipos de enfermedades y que estuvieran clasificadas dependiendo de la plaga que la afecta. Una base de datos que tiene esta característica es la de Esgario de 2018, donde se encuentran 2105 imágenes de hojas de café[3], catalogadas en: Hojas sanas, minador de hojas, Roya, cercospora y phoma. Estas imágenes se encuentran clasificadas en un archivo adicional con las diferentes enfermedades que tienen las hojas. Estas imágenes fueron tomadas en un ambiente controlado donde las imágenes fueron captadas con fondo blanco y solo es visible una hoja a la vez.

Otra base de datos que se tomó como referencia es el trabajo realizado por Parraga-Alava, Cusme, Loor, & Santander, donde se tomaron 1560 imágenes de campo que tienen en primer plano una hoja. Ésta base de datos se encuentra categorizada con:

[3] Disponible en: https://github.com/esgario/lara2018

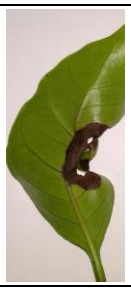
Hojas sanas, minador de hojas, Roya, cercospora, phoma y araña roja[4]. Una tercera base de datos que se encontró fue la elaborada por Francis Jesmar[5], la cual está conformada por 8.735 imágenes que fueron tomadas en campo y se encuentran clasificadas en: Hojas sanas, minador de hojas, Roya, cercospora, phoma y araña roja (Montalbo, 2022).

Al agrupar las bases de datos se completó una base de 12.419 imágenes con la que se procedió a realizar el entrenamiento de las redes neuronales, que serán comparadas para determinar cuál es la mejor opción al momento de implementarla en la aplicación móvil. En la siguiente tabla se puede observar un resumen de la composición de las diferentes bases de datos y ejemplos de las imágenes.

[4] Disponible en: https://data.mendeley.com/datasets/c5yvn32dzg/2
[5] Disponible en: https://github.com/francismontalbo/swatdcnn

Tabla 3

Cuadro comparativo imágenes de base de datos

Base de datos	Minador de hojas	Roya	Phoma	Cercospora	Arañita roja	Hoja sana
						
Esgario 2018	485	672	353	321	--	274
Parraga-Alava, Cusme, Loor, & Santander 2020	--	602	--	--	176	791
Francis Jesmar 2022	1275	1350	1263	1439	1368	2050
Total	1760	2624	1616	1760	1544	3115

Estas bases de datos son cargadas a Python por medio de la librería patlib donde se crea la ruta de acceso a la base de datos que se encuentra alojada en el disco duro del dispositivo con el fin de disminuir los tiempos de ejecución. Los códigos correspondientes a esta sección son los siguientes:

```
from pathlib import Path
import pathlib
data_dir='C:/bases/Proyecto/leafV2'
data_dir = pathlib.Path(data_dir)
```

4.2 Momento 2: Entrenamiento de red neuronal

En el proceso de entrenamiento es necesario la creación de un dataset, el cual se genera a partir de las imágenes que se tienen en la base de datos. Con el fin de reducir los tiempos de entrenamiento se usa la base de datos de Esgario. Para poder usar las imágenes durante el entrenamiento se usa el siguiente comando:

tf.keras.utils.image_dataset_from_directory con su respectiva estructura. Las imágenes deben estar clasificadas en carpetas de tal forma que la cantidad de carpetas y sus nombres correspondan a las categorías (plagas). La estructura de la función usada es la siguiente:

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

Donde:

Data_di:, es la dirección del contenedor de la base de datos

validation_split: define la fracción de las imágenes que serán utilizadas para el entrenamiento y validación. Para el caso se toma 0.2, correspondiente al 20% para validación y 80% para entrenamiento.

subset: conjunto de datos que se toman como referencia para en entrenamiento. Puede ser "training", “validation” o “both”.

seed: semilla aleatoria para el inicio de las transformaciones.

image_size: tamaño de la imagen para estandarizar la lectura

batch_size: tamaño de los lotes de entrenamiento.

4.2.1 Red neuronal densa

El primer modelo que se va a aprobar es una red neuronal con capas densas. Este modelo es construido con la instrucción *layers.Dense*, donde se establece el número de

neuronas. La primera capa tiene una cantidad de neuronas igual a la cantidad de datos de la imagen. Ya que la imagen que está siendo procesada es del tipo RGB, tiene 3 veces el ancho por el largo; la capa de salida tiene una neurona por categoría identificada. Para la activación que va a usar la activación “relu”, la cual activa la capa en caso de tener valores de superiores a cero, la activación es de forma lineal. La estructura de la creación del modelo es el siguiente:

```
num_classes = len(class_names)

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Flatten(),
  layers.Dense(512, activation='relu'),
  layers.Dense(512, activation='relu'),
  layers.Dense(512, activation='relu'),
  layers.Dense(num_classes)
])
```

El modelo constructivo de la red neuronal es el siguiente:

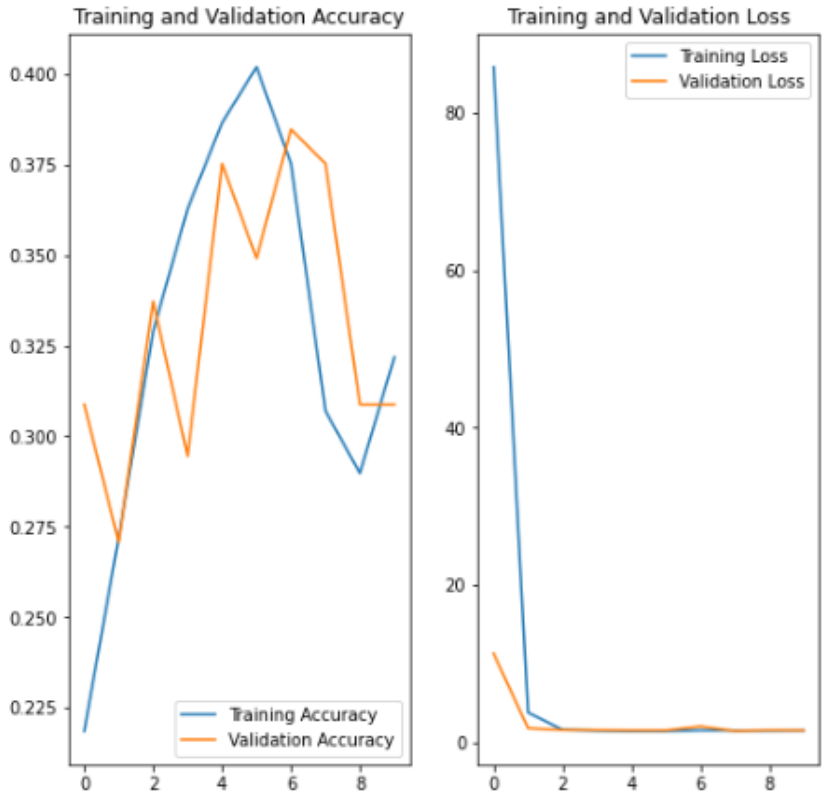
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_2 (Rescaling)     (None, 256, 512, 3)       0

 flatten_1 (Flatten)         (None, 393216)            0

 dense_2 (Dense)             (None, 512)               201327104

 dense_3 (Dense)             (None, 512)               262656

 dense_4 (Dense)             (None, 512)               262656

 dense_5 (Dense)             (None, 5)                 2565

=================================================================
Total params: 201,854,981
Trainable params: 201,854,981
Non-trainable params: 0
_________________________________________________________________
```

Una vez realizado el entrenamiento con solo 10 épocas, se obtiene las siguientes gráficas en las que se pueden visualizar los resultados de entrenamiento y de prueba.

En esta gráfica se puede interpretar que la precisión del modelo (Figura 14 – Derecha) tiende a mejorar en las primeras épocas, pero sufre pérdidas en las siguientes, dando a entender que el modelo no mejora sustancialmente con el trascurrir de las épocas, y las pérdidas (Figura 14 - izquierda) se mantienen un valor alto, siendo la primera de 85 y las siguientes de 1.5, tal como se muestra en la Figura 14. Por lo anterior, se deduce que el modelo no se acopla adecuadamente a la solución del problema debido a la simplicidad de las funciones que lo constituyen.

Figura 14

Resultado de entrenamiento red neuronal densa de 3 capas



4.2.2 Red neuronal Convolucional

El segundo modelo neuronal que se implementa es el modelo neuronal convolucional, en el cual se usa la sentencia *layers.Conv2D*, la cual crea el núcleo para la convolución y tiene como parámetros de entrada la cantidad de filtros, el ancho de la ventana de convolución, el relleno de la matriz de convolución, y la activación de la capa neuronal. Ésta sentencia da como salida un vector que en caso de ser activo, agrega el sesgo a las capas. Luego de realizar la convolución, es necesario agrupar los datos, lo cual se hace mediante la sentencia *layers.MaxPooling2D.* Este proceso se puede repetir cuantas veces se quiera aumentando la cantidad de filtros para tener mejoras en el resultado. Al finalizar, es necesario aplanar las capas mediante la sentencia *layers.Flatten* y agrupar mediante una capa densa con una neurona por cada clase, es decir una salida por cada plaga que está en la base de datos. El código completo que se implementó es el siguiente:

```
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

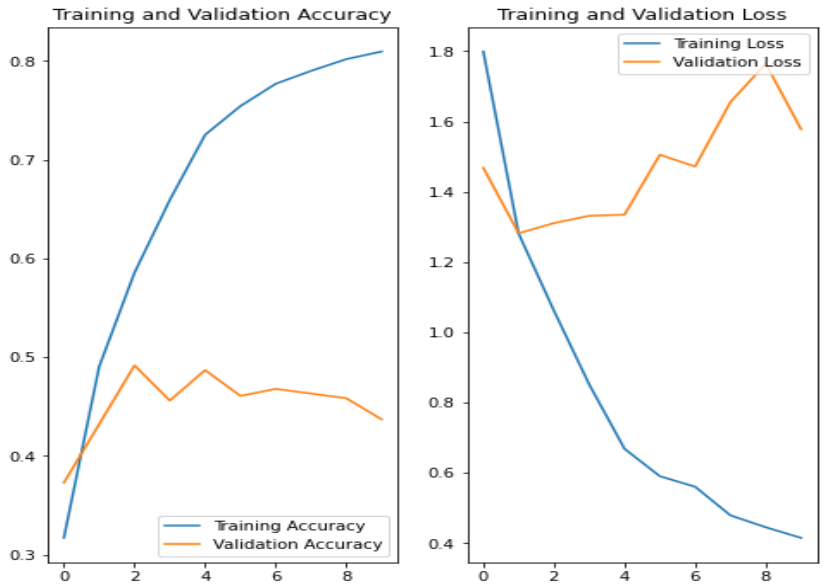
El modelo constructivo de la red neuronal es el siguiente:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 256, 512, 3)       0

 conv2d_3 (Conv2D)           (None, 256, 512, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 128, 256, 16)     0
 2D)

 conv2d_4 (Conv2D)           (None, 128, 256, 32)      4640

 max_pooling2d_4 (MaxPooling  (None, 64, 128, 32)      0
 2D)

 conv2d_5 (Conv2D)           (None, 64, 128, 64)       18496

 max_pooling2d_5 (MaxPooling  (None, 32, 64, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 131072)            0

 dense_2 (Dense)             (None, 128)               16777344

 dense_3 (Dense)             (None, 5)                 645

=================================================================
Total params: 16,801,573
Trainable params: 16,801,573
Non-trainable params: 0
_________________________________________________________________
```

El resultado del entrenamiento con 10 épocas se encuentra en la Figura 15, donde podemos ver que durante el entrenamiento se mejora la exactitud del modelo, pero con datos de validación la exactitud se mantiene por debajo de 0.5 (Figura 15 - derecha), mientras que las pérdidas del modelo durante el entrenamiento van disminuyendo desde 1.8 a 1.4 las perdidas en la validación sen mantienen en entre 1.25 y 1.7 (Figura 15 - izquierda), lo que indicaría que el modelo está aprendiendo de las imágenes del entrenamiento pero que no es capaz de reconocer los patrones globales, es decir, al analizar imágenes que no son del conjunto de datos de entrenamiento estas son categorizadas con un valor de precisión no muy alto al rededor del 0.8, por lo que el modelo no es óptimo para la aplicación que se desarrolla.

Figura 15

Resultado de entrenamiento red neuronal convolucional de 3 capas



4.2.3 Red neuronal Convolucional con sobreajuste (Overfitting)

Una forma de mejorar la red neuronal convolucional es impedir el overfitting o sobreajuste, el cual se presenta cuando el modelo aprende a predecir unos datos, pero no es capaz de generalizar las características ni de identificarlas en nuevos datos. Para solucionar esto, se implementa un método consistente en tomar los datos de validación y ajustar diferentes hiperparámetros. Para implementar esta solución, se usa en TensorFlow la sentencia *data_augmentation*, el cual se encarga de realizar transformaciones que resulten realistas. El modelo es el siguiente:

```
model2 = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])
```

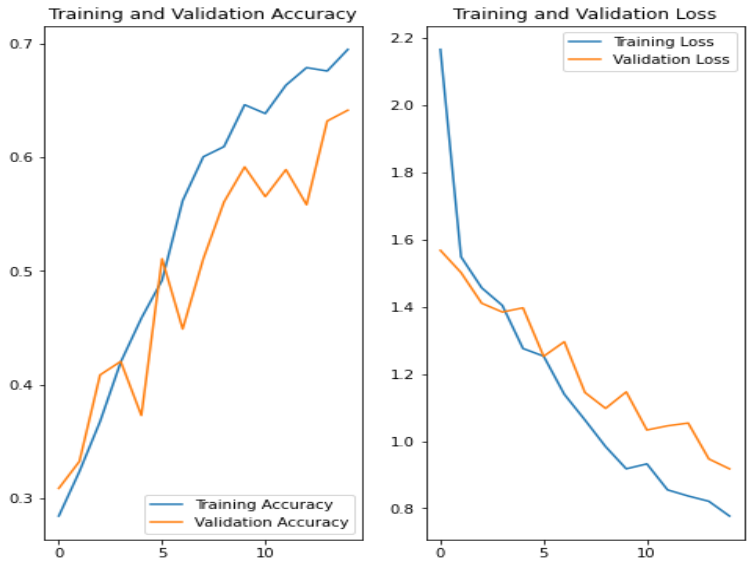
El modelo constructivo de la red neuronal con overfitting sería el siguiente, donde si lo comparamos con el modelo sin overfitting es el mismo, lo que cambia en el proceso de entrenamiento del modelo.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_1 (Sequential)   (None, 256, 512, 3)       0

 rescaling_2 (Rescaling)     (None, 256, 512, 3)       0

 conv2d_3 (Conv2D)           (None, 256, 512, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 128, 256, 16)     0
 2D)

 conv2d_4 (Conv2D)           (None, 128, 256, 32)      4640

 max_pooling2d_4 (MaxPooling  (None, 64, 128, 32)      0
 2D)

 conv2d_5 (Conv2D)           (None, 64, 128, 64)       18496

 max_pooling2d_5 (MaxPooling  (None, 32, 64, 64)       0
 2D)

 dropout (Dropout)           (None, 32, 64, 64)        0

 flatten_1 (Flatten)         (None, 131072)            0

 dense_2 (Dense)             (None, 128)               16777344

 outputs (Dense)             (None, 6)                 774

=================================================================
Total params: 16,801,702
Trainable params: 16,801,702
Non-trainable params: 0
_________________________________________________________________
```

Como resultado del entrenamiento podemos ver en la Figura 16 que al comparar los datos de entrenamiento y validación se observa que ambos siguen una misma tendencia, es decir, que el modelo ha aprendido a predecir datos externos a partir de los datos de entrenamiento por lo que ha tenido menores pérdidas en las predicciones (Figura 16 - izquierda), y la exactitud del modelo va mejorando con cada época (Figura 16 - derecha), lo que indica que podríamos implementar este modelo en la aplicación. Es necesario en un siguiente paso establecer cuál sería el número de épocas necesarias para un adecuado entrenamiento.

Figura 16

Resultado de entrenamiento red neuronal densa de 3 capas



4.3 Momento 3: Optimización de modelo neuronal

Luego de determinar qué modelo neuronal tiene mejores predicciones, se realiza una optimización del modelo. Para ello se utiliza como base el modelo neuronal convolucional con sobre ajuste y la base de datos de 12419 imágenes. Y se debe entrenar por varias

épocas, hasta obtener un valor de precisión o perdidas ya sea en el entrenamiento o en la validación que se considere estable para lo cual se usa la función *EarlyStooping*, la cual va comprando uno de estos parámetros y en específico en un determinado número de épocas consecutivas, y en caso de que se encuentre este parámetro valor dentro de una variación especifico, se considera que el modelo finaliza su entrenamiento. En caso de que no se llegase a cumplir la anterior condición, el entrenamiento culmina después de un cierto número de épocas. La implementación del código es la siguiente:

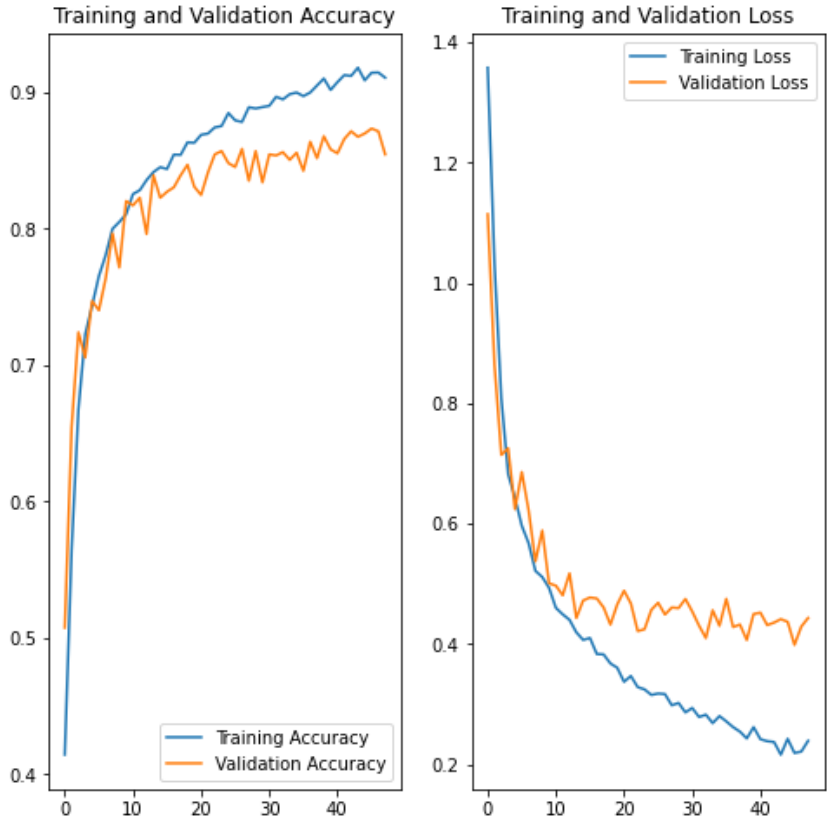
```
early_stop = tf.keras.callbacks.EarlyStopping(min_delta=0.001 ,monitor='accuracy',patience=4)
Epochs=100

history =model.fit(
    train_ds,
    epochs=Epochs,
    validation_data=val_ds,
    verbose=1,
    callbacks=[early_stop]
)
```

Para el entrenamiento de esta red, se toma en cuenta el parámetro accuracy y un delta de cambio máximo de 0.001 para 4 épocas del entrenamiento, luego de realizar múltiples entrenamientos donde se obtiene que es una variación donde el modelo realiza una adecuada predicción y ha llegado a estabilizar la predicción. Como resultado se obtiene que el modelo tardó 48 épocas para llegar a esta condición con los siguientes valores en la época 48: perdidas: 0.2394 – exactitud: 0.9108 – perdidas en validación: 0.4436 – exactitud en validación: 0.8546. Las gráficas de precisión y pérdidas del modelo son las mostradas en la Figura 17, donde se puede apreciar que las pérdidas (Figura 17 - derecha) en el entrenamiento son menores a la de las validaciones, las cuales son de 0.5, mejorando los modelos anteriores. La precisión del modelo con datos de entrenamiento es superior al 0.9 y con datos de validación es superior a 0.81 (Figura 17 - izquierda), con lo que se obtiene un modelo más preciso que al ser utilizado en la

predicción de las enfermedades, nos da una predicción de las plagas que se están intentado identificar.

Figura 17

Resultado de entrenamiento red neuronal convolucional de 3 capas, con sobreajuste y EarlyStopping.



4.4 Momento 4: Diseño de aplicación para Android

Una vez entrenado el modelo con los valores de predicción deseados, se desarrolla la aplicación java en Android Studio, en la cual se importa el modelo neuronal implementado con la biblioteca de TensorFlow Lite.

La aplicación contiene las siguientes secciones:

- Un SplashScreen dónde se presenta un logo de la aplicación.
- La pantalla de Identificación donde se realizará la identificación de las plagas.
- La pantalla de información de la aplicación

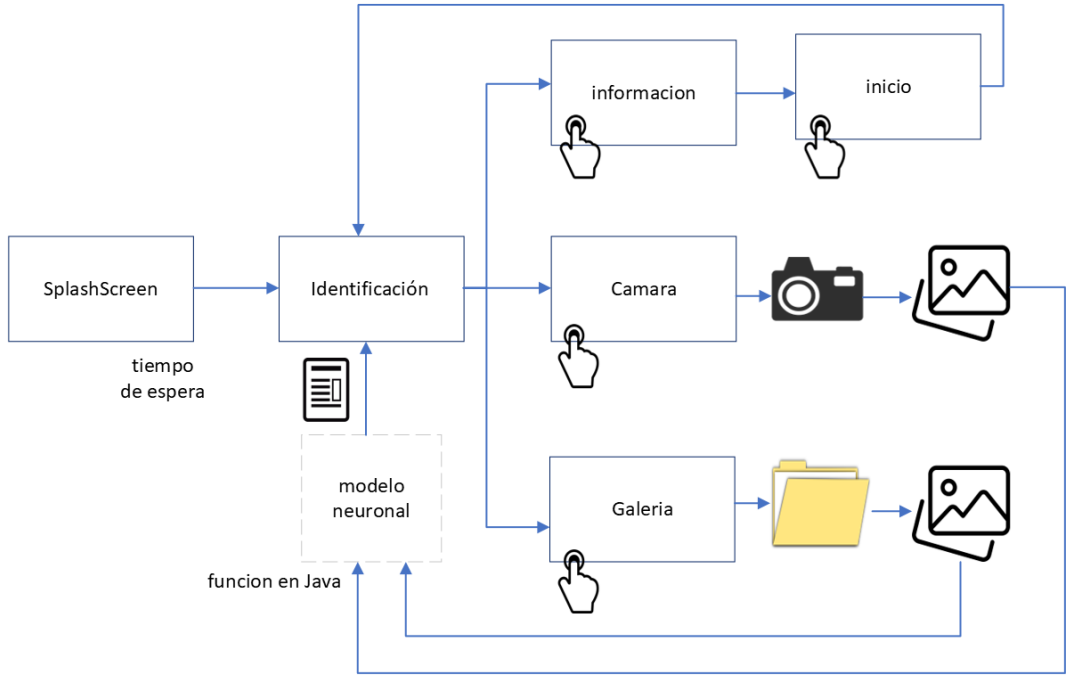
La pantalla de identificación cuenta con 3 botones, el primer botón que lleva a la pantalla de información básica de la aplicación, el segundo botón llamado a Cámara, el cual inicia la función de Cámara y entrega como resultado una imagen, y el tercer botón accede a la galería, la cual permite navegación en los archivos del dispositivo y entrega como resultado la imagen para ser procesada.

Cuando la imagen se adquiere, ya sea por cámara o galería, esta se escala y se lleva al modelo de red neuronal por medio de un buffer en Java; usando las librerías de TensorFlow Lite se obtiene la predicción sobre cuál es la plaga que está afectando el cultivo y este resultado se imprime en un texto en la pantalla de identificación.

En la figura 18 se puede ver cómo es la navegación en la aplicación, iniciando con la pantalla del SplashScreen, donde la aplicación espera 2 segundos, y carga la página de identificación, donde por medio de botones el usuario elige si cargar una imagen desde la cámara, desde la galería, o ver la información de la aplicación.

Figura 18

Diagrama de flujo de la aplicación Android.



4.4.1 Diseño del logo

El diseño del logo de la aplicación se hace con ayuda de la inteligencia artificial: Dali-E2[6], con las palabras claves cafeto y plagas. Esta imagen se puede cargar al proyecto en Android Studio como *Image Asset* y en donde se ajusta para diferentes vistas del logo. El resultado se encuentra en la Figura 19.

[6] https://openai.com/dall-e-2/

Figura 19

(a) Logo desarrollado con Dali-E2 para la aplicación, (b) implementación como logo de la aplicación

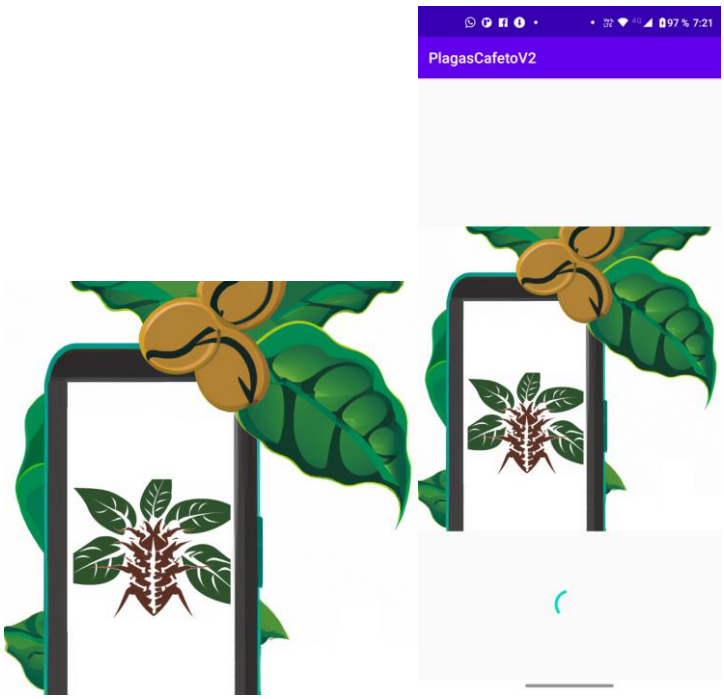
4.4.2 Diseño de SplashScreen

El diseño del SplashScreen de la aplicación se hace con ayuda de la inteligencia artificial: Dali-E2[7], con las palabras claves cafeto, plagas y aplicación para celular, su implementación se hace por medio de un imageView, donde se carga en la pantalla la imagen por 2 segundos y se acompaña con un progresBar, el cual a medida que tiene una animación durante el tiempo que dure la vista de la pantalla activa. La implementación del SplashSreen es la siguiente:

[7] https://openai.com/dall-e-2/

Figura 20

(a) SplashScreen desarrollado con Dali-E2 para la aplicación, (b) utilización en la aplicación



4.4.3 Pantalla de identificación

Para identificar las plagas se tiene una interfaz en la que el usuario puede elegir si carga imágenes desde la galería o toma una foto con el dispositivo. También se dispone de un botón que direcciona a una pantalla con información sobre la aplicación, un textView que permite imprimir en pantalla el resultado de la predicción y un ImageView que muestra la imagen que entra en el modelo neuronal. La pantalla de identificación es la siguiente:

Figura 21

Interfaz ventana de identificación de plagas

4.4.4 Botón de información

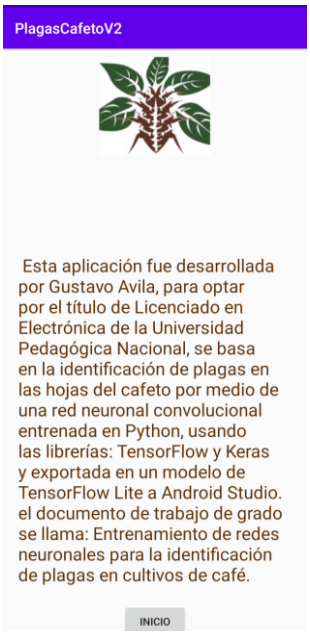
Este botón lleva una ventana con la información básica sobre la aplicación. Está programado en Java por medio de las funciones, findViewById y onClick, las cuales identifican que el botón es activado e inician la función que muestra la información. El código de programación es el siguiente:

```
bntInfo = findViewById(R.id. ButtonInfo); //Boton informacion //
bntInfo.(new View.OnClickListener() {
 @Override
 public void onClick(View view) {
 Intent intent = new Intent(IdentifyActivity.this, InforActivity.class);
 startActivity(intent);
 }

});
```

Esta ventana de interfaz de información contiene: el logo e información sobre la finalidad, el desarrollador, las librerías y el lenguaje que fue utilizado para el entrenamiento de la red neuronal, el usuario vería la información de la siguiente forma:

Figura 22

Interfaz ventana de información sobre la aplicación



4.4.5 Botón de cámara

Este botón, al igual que el botón de información, está conformado por la función que lo está escuchando constantemente y una función que se ejecuta cuando es obturado y que abre la cámara. La función de abrir cámara se ejecuta por medio del objeto intent, el cual ejecuta funciones que se están implementando en simultaneo, en este caso llama a los recursos de hardware por medio de la ejecución de la MediaStore que permite tomar una imagen de la cámara.

```
//boton camara//
btnCamera = findViewById(R.id.ButtonCamera);
Camera_View = findViewById(R.id.CaptureImage);
```

```
btnCamera.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View view) {

 abrirCamara();
 }
});


private void abrirCamara() {
 Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
 startActivityForResult(intent, 2);

}
```

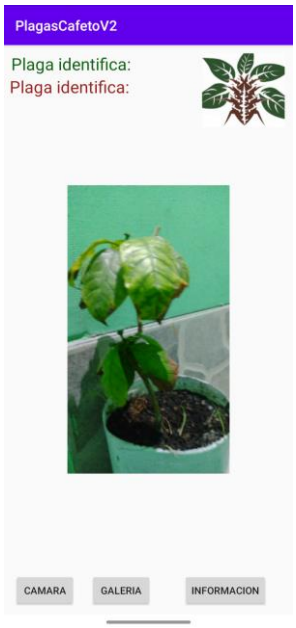
Para poder visualizar los resultados es necesaria la información que entrega la cámara para generar un Bitmap y así poder visualizar la imagen en la pantalla de identificación, esto se logra mediante el siguiente código:

```
Bitmap imgBitmap = (Bitmap) data.getExtras().get("data");
int dimension = Math.min(imgBitmap.getWidth(), imgBitmap.getHeight());
imgBitmap = ThumbnailUtils.extractThumbnail(imgBitmap, dimension, dimension);
Camera_View.setImageBitmap(imgBitmap);
```

El resultado de la captura de imagen a través de la Cámara es el siguiente, donde se puede observar la imagen sobre la cual se quiere hacer la predicción.

Figura 23

Interfaz ventana de identificación con imagen de cámara



4.4.6 Botón de galería

Este botón de galería, al igual que los anteriores, contiene una función que lo está escuchando constantemente y al ser obturado llama a otra función llamada External_content_URL por medio de Media Store, la cual hace una navegación a través de la galería y entrega un url del contenedor de imagen. Esto se hace por medio del siguiente código:

```
btnGaleria = findViewById(R.id.ButtonGaleria);

btnGaleria.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View view) {
 Intent intent = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
 intent.setType("image/");
 startActivityForResult(intent, 3);
 }
});
```

Para poder imprimir el contenido del url es necesario hacer una transformación a Bitmap, lo cual también nos ayuda con la exportación de esta imagen al modelo neuronal. El código es el siguiente:

```
Uri dat = data.getData();
Bitmap imgBitmap = null;

try {
 imgBitmap = MediaStore.Images.Media.getBitmap(this.getContentResolver(),
dat);
} catch (IOException e) {
 e.printStackTrace();
}
Camera_View.setImageBitmap(imgBitmap);
```

El resultado de la captura de imagen a través de la galería es el siguiente:

Figura 24

Interfaz ventana de identificación con imagen de galería

4.4.7 importar modelo de TensorFlow Lite

El modelo creado en TensorFlow es importado mediante Android Studio en un repositorio ml. Los códigos base para trabajar con el modelo son generados de forma

automática, pero es necesario implementar un buffer para la conexión del modelo neuronal con la aplicación en Java, realizar la lectura del resultado y etiquetarla según sean las categorías de entrenamiento.

```
Model1v4 model = Model1v4.newInstance(getApplicationContext());

model.close();
```

4.4.8 Buffer entre Imagen y Modelo de TensorFlow Lite

Una vez se carga la imagen al visualizador, esta debe ser llevada al modelo TensorFlow. Para ello es necesario usar un TensorBuffer, que crea un arreglo de 3 veces el tamaño de la imagen, ya que la imagen está compuesta por pixeles RGB, para ser cargado el archivo de la imagen, que se encuentra en un Bitmap, a la red neuronal. Esto se hace mediante el siguiente código:

```
TensorBuffer inputFeature0 = TensorBuffer.createFixedSize(new int[]{1,
imagensizeY, imagesizeX, 3}, DataType.FLOAT32);
ByteBuffer byteBuffer =
ByteBuffer.allocateDirect(4*imagensizeY*imagesizeX*3);
byteBuffer.order(ByteOrder.nativeOrder());

int[] intValues = new int[imagensizeY * imagesizeX];
image.getPixels(intValues, 0, image.getWidth(), 0, 0, image.getWidth(),
image.getHeight());
int pixel = 0;

for(int i = 0; i < imagensizeY; i ++){
 for(int j = 0; j < imagesizeX; j++){
 int val = intValues[pixel++]; // RGB
 byteBuffer.putFloat(((val >> 16) & 0xFF) * (1.f / 1));
 byteBuffer.putFloat(((val >> 8) & 0xFF) * (1.f / 1));
 byteBuffer.putFloat((val & 0xFF) * (1.f / 1));
 }
}

inputFeature0.loadBuffer(byteBuffer);
```

Una vez el modelo neuronal es cargado con la imagen, este se ejecuta y entrega una predicción por cada categoría en el entrenamiento. Para identificar cual es la plaga que probablemente está afectando más las hojas, se implementa el siguiente código donde

se selecciona la salida de mayor valor, en caso de que la probabilidad sea negativa se establece en 0.

```
// Runs model inference and gets result.
Model1v4.Outputs outputs = model.process(inputFeature0);
TensorBuffer outputFeature0 = outputs.getOutputFeature0AsTensorBuffer();

float[] confidences = outputFeature0.getFloatArray();
int maxPos = 0;
float maxConfidence = 0;
for (int i = 0; i < confidences.length; i++) {
 if (confidences[i] > maxConfidence) {
 maxConfidence = confidences[i];
 maxPos = i;
 }
      if (confidences[i] < 0) {
      confidences[i] = 0;
      }

}
```

Para poder mostrar los resultados de forma más precisa y poder comparar las posibles afectaciones, se muestran los resultados de las 6 neuronas de salida en una lista de textos donde se presenta la probabilidad que tiene cada plaga de ser la que está afectando el cultivo, lo cual es posible mediante el siguiente código:

```
String[] classes = {"Araña Roja", "Hoja Sana", "Minador De Hojas", "Roya",
"cercospora", "phoma"};
result0.setText("Plaga identificada: " + classes[maxPos] );
result1.setText(classes[0] +" Probabilidad: " + (confidences[0]*10 + "%"));
result2.setText(classes[1] +" Probabilidad: " + (confidences[1]*10 + "%"));
result3.setText(classes[2] +" Probabilidad: " + (confidences[2]*10 + "%"));
result4.setText(classes[3] +" Probabilidad: " + (confidences[3]*10 + "%"));
result5.setText(classes[4] +" Probabilidad: " + (confidences[4]*10 + "%"));
result6.setText(classes[5] +" Probabilidad: " + (confidences[5]*10 + "%"));
```

Como resultado de la aplicación se puede observar las siguientes imágenes:

Figura 25

Resultados de la identificación de plagas (a) Araña Roja, (b) Roya, (c) Roya

Capítulo 5

5.1 Conclusiones, Recomendaciones y Trabajo Futuro

En esta sección se comentan los resultados obtenidos en el desarrollo del proyecto, algunas recomendaciones y posibles trabajos a futuro a partir de este trabajo.

- Se da cumplimiento a los objetivos específicos en la identificación de las plagas en el café por medio de la implementación de visión artificial, y el reconocimiento de imágenes de las hojas de la planta.
- Se implementa una interfaz gráfica en Android donde es posible realizar la identificación de las plagas por parte del usuario de la aplicación.
- Se cumple el objetivo general del proyecto al tener la identificación por visión artificial y la aplicación de reconocimiento de imagen para Android funcionado simultáneamente.

- La identificación realizada por la red neuronal responde de forma adecuada a los valores de validación y corresponde adecuadamente a la identificación de plagas en terreno.
- Los modelos de redes neuronales están siendo aplicados en visión e inteligencia artificial para resolver varios tipos de problemas, tanto de clasificación como predicción de datos, por eso es necesario profundizar en el estudio y aplicación de las redes neuronales, con el fin de poder atender a los nuevos requerimientos del mercado y poder implementar en el aula de clase la implementación y estudio de redes neuronales.

Con respecto al funcionamiento de la aplicación se puede observar que:

- La probabilidad al identificar la plaga se pueden mostrar valores altos debido a la calidad de la imagen, al proceso que se realiza para escalar la misma, al ser un modelo donde las bases de datos se enfocan en una sola hoja tener múltiples hojas en la imagen genera predicciones simultaneas.
- El lenguaje Java, utilizado en este proyecto ofrece gran cantidad de herramientas para la implementación de aplicaciones en Android. Sin embargo, el lenguaje Kotlin puede ser una alternativa adicional teniendo en cuenta su amplia incursión en el mercado de las aplicaciones para teléfonos móviles.
- El modelo convolucional con overfitting, presenta el mejor resultado de predicción, pero aun así, es un modelo limitado que podría ser remplazado por otros modelos de mayor alcance, como: ResNet50, CIFAR-10, VGG19.

- Al tener una cantidad de plagas tan limitada, es evidente que no se puede hacer un reconocimiento más detallado de las plagas, por lo que sería necesaria ampliar la base de datos.

Como recomendación para trabajos futuros en redes neuronales, es necesario ampliar la base de datos de enfermedades y cantidad de muestras para tener mejores resultados en la predicción de datos. Experimentar con arquitecturas más complejas de redes neuronales que permitan también la identificación de múltiples objetos en la misma fotografía.

En términos de aplicación móvil, se recomienda el uso de Kotlin por su creciente uso en los últimos años, ya que, en comparación con Java, presenta más versatilidad, confianza, seguridad y estabilidad al ejecutar aplicaciones, permitiendo incluso ejecutar el modelo neuronal mientras se ejecuta la aplicación de la cámara en simultáneo.

Referencias

Bravo Durán, V., de la Cruz Malavassi, E., Herrera Ledezma, G., & Ramírez Muñoz, F. (Enero – junio 2013). Uso de plaguicidas en cultivos agrícolas como herramienta para el monitoreo de peligros en salud. *UNICIENCIA*, 351-376.

Alvarez Germade, Y., Barbará Morales, E., & Rodriguez Ramirez , O. (2010). FILTRADO DIGITAL EN EL PROCESAMIENTO DE IMÁGENES EMPLEANDO MATLAB. *Convención Científica de Ingeniería y Arquitectura*.

Baird, D. (1991). El principio de mínimos cuadrados . En *En Experimentación: Una introducción a la teoría de mediciones y al diseño de experimentosa la teoría de*

mediciones y al diseño de experimentos (págs. 172-177). México: Pearson Prentice Hal.

BENAVIDES M, P. G. (2013). Plagas del café Broca, minador, cochinillas harinosas arañita roja y monalonion. *Manual cafetera colombiano: investigación y tecnología para la sostenibilidad de la caficultura*, 2016-225.

Bertona, L. (2005). *Entrenamiento de Redes Neuronales Basado en Algoritmos Evolutivos.*

Bustillo Pardey, Á. E. (2007). *El manejo de cafetales y su relación con el control de la broca del café en Colombia.* Manizales : CENICAFÉ.

Casa Robles, P. C. (2020). *Desarrollo de aplicaciones móviles de clasificación y detección de objetos a partir de redes convolucionales ligeras (Master's thesis).*

Constantino, L. M. (2011). Aspectos biológicos, morfológicos y genéticos de Hypothenemus obscurus e Hypothenemus hampei (Coleoptera: Curculionidae: Scolytinae). *Revista Colombiana de Entomología*, 173-18.

Constantino, L. M. (2013). *Minador de las hojas del cafeto: Una plaga potencial por efectos del cambio climático.* Manizales : Centro Nacional de Investigaciones de Café (Cenicafé).

Del Puerto Rodríguez, A., Suárez Tamayo, S., & Palacio Estrada, D. (2104). Efectos de los plaguicidas sobre el ambiente y la salud. *Revista Cubana de Higiene y Epidemiología*, 372-387.

Dietterich, T. (1995). Overfitting and undercomputing in machine learning. *ACM computing surveys*, 326-327.

Enriquez, E. B. (1975). Morfología, Ciclo Biológico y Comportamiento de Leucoptera coffeella Guer-Men. *Revista Peruana de Entolomogía*, 79-81.

Esgario, J. G. (2020). Deep learning for classification and severity estimation of coffee leaf biotic stress. *Computers and Electronics in Agriculture*.

GIL, Z. C. (2013). *Aprenda a manejar la arañita roja del café.* Cenicafé.

Gil-Palacio, Z. C.-M. (2021). Diagnóstico de las cochinillas de las raíces del café en ocho departamentos cafeteros de Colombia. *Avances técnicos Cenicafé*, 1-8.

Gil-Vallejo, L. F., & LeguizamÛn-Caycedo, J. E. (2000). *LA MUERTE DESCENDENTE DEL CAFETO (Phoma spp.).* Chinchina : Cenicafe.

Hernández-Martínez, G. &.-P. (2016). Análisis integral sobre la roya del café y su control. *RINDERESU*, 92-99.

Instituto Colombiano Agropecuario (ICA). (07 de Febrero de 2014). *Para combatir el picudo de la guayaba productores deben adoptar medidas fitosanitarias eficientes*. Obtenido de Noticias Instituto Colombiano Agropecuario : https://www.ica.gov.co/noticias/agricola/2013-(1)/para-combatir-el-picudo-de-la-guayaba-productores

J. G. M. Esgario, R. A. (2019). Deep Learning for Classification and Severity Estimation of Coffee Leaf Biotic Stress. *Computers and Electronics in Agriculture*, 169. Obtenido de https://arxiv.org/abs/1907.11561

Jaramillo, M. G., Benavides machado, p., & Constantino, L. M. (2015). *Conozca al pasador de las ramas del café : Un insecto plaga ocasional en Colombia.* Manizales: CENICAFE.

Keras. (30 de 11 de 2022). *Keras, Simple. Flexible. Poderoso.* Obtenido de https://keras.io/

Marín-Ramírez, G. L.-V.-F. (2019). *Alertas tempranas para el manejo de enfermedades.* Cenicafé.

Martín Abadi, A. A. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems,*. Obtenido de Software available from tensorflow.org.: ww.tensorflow.org

Matich, D. J. (2001). Informática Aplicada a la Ingeniería de Procesos – Orientación I. *Redes Neuronales: Conceptos Básicos y Aplicaciones.* Rosario - Mexico: Universidad Tecnológica Nacional.

Montalbo, F. J. (2022). Automated diagnosis of diverse coffee leaf images through a stage-wise aggregated triple deep convolutional neural network. *Machine Vision and Applications*, 33(1), 1-22.

Müller , A. C., & Guido, S. (2018). *Introduction to Machine Learning with Python: a guide for data scientists.* " O'Reilly Media, Inc.".

Muñoz, O. A. (2019). *Diseño de un sistema de visión artificial para el análisis de calidad y producción de rosas.* Obtenido de http://hdl.handle.net/20.500.12209/12074.

NumPy. (2022). *NumPy documentación* . Obtenido de https://numpy.org/doc/stable/

Oliveira, A. J. (2019). Analysis of nematodes in coffee crops at different altitudes using aerial images. *27th European Signal Processing Conference (EUSIPCO)*, 1-5.

Parraga-Alava, J., Cusme, K., Loor, A., & Santander, E. (2019). RoCoLe: A robusta coffee leaf images dataset. *Mendeley Data V2*. doi:10.17632/c5yvn32dzg.2

Python Software Foundation. (26 de 12 de 2022). *Historia y Licencia*. Obtenido de Historia y Licencia: https://docs.python.org/es/3/license.html

Reddy, K. T. (2017). Técnicas de procesamiento de imágenes para la detección de formas de insectos en cultivos de campo. *Conferencia internacional sobre computación e informática inventivas (ICICI) de 2017*, 699-704.

Rengifo, H. G. (202). *Algunos aspectos biológicos de Cercospora coffeicola.*

Rivillas, C. A. (2011). *La roya del cafeto en Colombia: Impacto manejo y costos del control.* Chinchiná: Cenicafe.

Roman Gonzalez, A. (2012). Análisis de imágenes digitales. *ECIPeru*, 61-68.

Sánchez-Méndez, A. G.-H. (2018). Análisis de imágenes multiespectrales para la detección de cultivos y detección de plagas y enfermedades en la producción de café. *Res. Comput. Sci*, 309-317.

TensorFlow. (1 de 12 de 2022). *TensorFlow Lite*. Obtenido de https://www.tensorflow.org/lite/guide

Torres, J. (2020). *Python deep learning: introducción práctica con Keras y TensorFlow 2.* Marcombo.

Valencia, J. H. (2022). Construcción de una app nativa Android para la detección facial de emociones usando técnicas de inteligencia artificial. *Pro Sciences: Revista de Producción, Ciencias e Investigación*, 52-61.

Villegas, C. &. (2014). *Identificación de cochinillas harinosas en las raíces de café en departamentos cafeteros de Colombia.*

Villegas, C. P. (2015). *Aspectos del ciclo de vida de Puto barberi Cockerell Hemiptera: Putoidae.*

Villegas, C. Z. (2010). *Identificación y hábitos de cochinillas harinosas asociadas a raíces del café en Quindío.*

Vorobioff, J., Cerrotta, S., Eneas Morel, N., & Amadio, A. (20222). *Inteligencia Artificial y Redes Neuronales: Fundamentos Ejercicios y Aplicaciones con Phyton y Matlab.* Buenos Aires: edUTecNe.