UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UNA APLICACIÓN MÓVIL PARA LA DETECCIÓN DE ENFERMEDADES EN LA MAZORCA DE CACAO UTILIZANDO DEEP LEARNING.

RUILOVA CUMBICOS RAUL VINICIO
INGENIERO DE SISTEMAS

MACHALA
2022

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UNA APLICACIÓN MÓVIL PARA LA DETECCIÓN DE ENFERMEDADES EN LA MAZORCA DE CACAO UTILIZANDO DEEP LEARNING.

RUILOVA CUMBICOS RAUL VINICIO
INGENIERO DE SISTEMAS

MACHALA
2022

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

TRABAJO TITULACIÓN
PROPUESTAS TECNOLÓGICAS

DESARROLLO DE UNA APLICACIÓN MÓVIL PARA LA DETECCIÓN DE ENFERMEDADES EN LA MAZORCA DE CACAO UTILIZANDO DEEP LEARNING.

RUILOVA CUMBICOS RAUL VINICIO
INGENIERO DE SISTEMAS

RIVAS ASANZA WILMER BRAULIO

MACHALA, 23 DE FEBRERO DE 2022

MACHALA
2022

Tesis raul tuilova

INFORME DE ORIGINALIDAD

6%	5%	1%	1%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	dspace.utb.edu.ec Fuente de Internet	1%
2	repositorio.ug.edu.ec Fuente de Internet	1%
3	www.slideshare.net Fuente de Internet	<1%
4	dspace.espoch.edu.ec Fuente de Internet	<1%
5	github.com Fuente de Internet	<1%
6	repositorio.cinvestav.mx Fuente de Internet	<1%
7	repositorio.ucundinamarca.edu.co Fuente de Internet	<1%
8	ribuni.uni.edu.ni Fuente de Internet	<1%
9	documentop.com Fuente de Internet	<1%

CLÁUSULA DE CESIÓN DE DERECHO DE PUBLICACIÓN EN EL REPOSITORIO DIGITAL INSTITUCIONAL

El que suscribe, RUILOVA CUMBICOS RAUL VINICIO, en calidad de autor del siguiente trabajo escrito titulado DESARROLLO DE UNA APLICACIÓN MÓVIL PARA LA DETECCIÓN DE ENFERMEDADES EN LA MAZORCA DE CACAO UTILIZANDO DEEP LEARNING., otorga a la Universidad Técnica de Machala, de forma gratuita y no exclusiva, los derechos de reproducción, distribución y comunicación pública de la obra, que constituye un trabajo de autoría propia, sobre la cual tiene potestad para otorgar los derechos contenidos en esta licencia.

El autor declara que el contenido que se publicará es de carácter académico y se enmarca en las dispociones definidas por la Universidad Técnica de Machala.

Se autoriza a transformar la obra, únicamente cuando sea necesario, y a realizar las adaptaciones pertinentes para permitir su preservación, distribución y publicación en el Repositorio Digital Institucional de la Universidad Técnica de Machala.

El autor como garante de la autoría de la obra y en relación a la misma, declara que la universidad se encuentra libre de todo tipo de responsabilidad sobre el contenido de la obra y que asume la responsabilidad frente a cualquier reclamo o demanda por parte de terceros de manera exclusiva.

Aceptando esta licencia, se cede a la Universidad Técnica de Machala el derecho exclusivo de archivar, reproducir, convertir, comunicar y/o distribuir la obra mundialmente en formato electrónico y digital a través de su Repositorio Digital Institucional, siempre y cuando no se lo haga para obtener beneficio económico.

Machala, 23 de febrero de 2022

Raul Ruilova

RUILOVA CUMBICOS RAUL VINICIO
0704396241

DEDICATORIA

El siguiente trabajo se lo dedico a mis padres, quienes me dieron la vida y con su apoyo incondicional, amor y sacrificio me han permitido llegar hasta este momento tan importante de mi formación profesional. A mis hermanos y hermana que siempre me han prestado su ayuda y motivado a seguir adelante. A mis sobrinos que son mi inspiración y siempre los llevo en mi corazón.

Finalmente dedicar a todas las personas que han extendido su mano de apoyo a lo largo de mi carrera universitaria y de mi vida.

Ruilova Cumbicos Raúl Vinicio

AGRADECIMIENTO

Agradezco a mis padres quienes han sido el motor y la guía fundamental en mi vida para conquistar mis objetivos y permitirme culminar mi carrera profesional. A mi familia en general que siempre han estado dispuestos a escucharme y apoyarme.

Así mismo, expresar mis agradecimientos a la escuela de Informática de la UTMACH, a mis profesores en especial a mi tutor, Ing. Sist. Rivas Asanza Wilmer Braulio, Phd gracias a sus conocimientos, dedicación y paciencia me han permitido crecer como profesional y culminar con este trabajo.

Ruilova Cumbicos Raúl Vinicio

RESUMEN

El cacao es susceptible a muchas enfermedades que pueden producir perdidas devastadoras, este es uno de los principales problemas para los pequeños y medianos productores, y hasta para los grandes. Por ello, es clave que en los cultivos de cacao se lleve un correcto control fitosanitario durante todo su ciclo de vida, principalmente en los primeros meses.

Las malas prácticas en los cultivos favorecen en gran medida a la aparición de enfermedades y como consecuencia a la pérdida de plantaciones. Las condiciones ambientales también pueden influir al desarrollo de las mismas cuando estas se encuentran en un estado prematuro.

Existen varias estrategias de prevención de enfermedades, sin embargo, ninguna de estas medidas asegura la eliminación completa de la afección en el cacao. Por ello es necesario adoptar un enfoque de control para identificar manualmente las mazorcas que han llegado a ser afectadas.

Para resolver este problema, el presente trabajo tiene como propósito desarrollar un modelo deep learning capaz de reconocer las enfermedades fitóftora y monilia en mazorcas de cacao. Debido a la capacidad de poder identificar varios objetos en una imagen se decidió elegir un algoritmo de detección de objetos como lo es YOLO en su versión más actual. El modelo será incorporado en una aplicación móvil como producto final, donde se podrá capturar las imágenes mediante la cámara o galería para ejecutar la detección.

El proceso de desarrollo se basó en primera instancia de la construcción de un dataset que contenga imágenes con mazorcas enfermas y sanas. Para ello se consideró la captura de fotos y la adición de un dataset encontrado en el repositorio Kaggle. Posteriormente se realizó el etiquetado de cada imagen mediante LabelImg, seleccionando las mazorcas con sus respectivas clases en un cuadro delimitador. Luego, se utilizó la plataforma Roboflow que permitió automatizar las tareas para que nuestro dataset posea el formato deseado.

El entrenamiento del modelo con YOLOv5 se llevó a cabo en la nube, mediante el servicio de Google Colaboratory para hacer uso de una GPU. El entrenamiento

se basó en la técnica de transfer learning para mejorar nuestro modelo final y obtener mejores resultados dada la limitación de nuestro dataset. El modelo final obtuvo valores de 0.80 de mAP, 0.88 de precisión y 0.73 de sensibilidad. Para utilizarlo en una aplicación móvil, el modelo fue convertido a un formato ONNX y luego con el framework Tencent NCNN a su respectivo formato.

La implementación de la aplicación móvil se llevó a cabo según los procedimientos establecidos en la metodología Mobile-D, por lo que se trabajó cada fase con iteraciones. La aplicación se desarrolló utilizando Android Studio y se conformó de dos módulos que son: detección y galería. Permitiendo así realizar la ejecución del modelo dentro del módulo de detección y la visualización de las imágenes resultantes en el módulo galería.

Para valorar el rendimiento de nuestro modelo, se realizaron las pruebas utilizando el conjunto test en la aplicación móvil. A partir de las pruebas se obtuvo una matriz de confusión para cada clase y las siguientes métricas: precisión del 0.85, sensibilidad del 0.84 y valor de referencia del 0.84. Con estos resultados se valida que nuestro modelo junto a la aplicación es capaz de detectar las mazorcas enfermas por fitóftora y monilia, además de las que no se encuentran afectadas mediante la clase sana.

Palabras clave: deep learning, detección de objetos, yolov5, enfermedades de cacao, aplicación móvil.

ABSTRACT

Cocoa is susceptible to many diseases that can cause devastating losses, this is one of the main problems for small and medium producers, and even for large ones. For this reason, it is key that cocoa crops carry out a correct phytosanitary control throughout their life cycle, mainly in the first months.

Bad practices in crops greatly favor the appearance of diseases and, as a consequence, the loss of plantations. Environmental conditions can also influence their development when they are in a premature state.

There are several disease prevention strategies, however, none of these measures ensures the complete elimination of the condition in cocoa. Therefore, it is necessary to adopt a control approach to manually identify the ears that have become affected.

To solve this problem, the present work aims to develop a deep learning model capable of recognizing phytophthora and monilia diseases in cocoa pods. Due to the ability to identify several objects in an image, it was decided to choose an object detection algorithm such as YOLO in its most current version. The model will be incorporated into a mobile application as a final product, where images can be captured through the camera or gallery to execute the detection.

The development process was based in the first instance on the construction of a dataset that contains images with sick and healthy ears. For this, the capture of photos and the addition of a dataset found in the Kaggle repository were considered. Subsequently, each image was labeled using LabelImg, selecting the ears with their respective classes in a bounding box. Then, the Roboflow platform was used, which allowed us to automate the tasks so that our dataset has the desired format.

The training of the model with YOLOv5 was carried out in the cloud, using the Google Collaboratory service to make use of a GPU. The training was based on the transfer learning technique to improve our final model and obtain better results given the limitation of our dataset. The final model obtained values of 0.80 for mAP, 0.88 for precision and 0.73 for recall. To use it in a mobile application,

the model was converted to an ONNX format and then with the Tencent NCNN framework to its respective format.

The implementation of the mobile application was carried out according to the procedures established in the Mobile-D methodology, so the phases were worked with iterations. The application was developed using Android Studio and was made up of two modules: detection and gallery. Thus allowing the execution of the model within the detection module and the display of the resulting images in the gallery module.

To assess the performance of our model, the tests were performed using the test set in the mobile application. From the tests, a confusion matrix was obtained for each class and the following metrics: precision of 0.85, recall of 0.84 and f-score of 0.84. With these results, it is validated that our model, together with the application, is capable of detecting cobs affected by phytophthora and monilia, in addition to those that are not affected.

Keywords: deep learning, object detection, yolov5, cocoa diseases, mobile application.

CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

INTRODUCCIÓN

Para Ecuador, el cacao se ha consolidado como uno de los productos más significativos gracias a la historia del país, desde el boom cacaotero donde se convirtió en una de las fuentes más importantes de ingresos sentando bases para la economía hasta la actualidad donde su calidad atrae el interés de los mercados internacionales.

Para los productores de cacao, el sobrellevar un cultivo conlleva una variedad de gastos en el establecimiento de la plantación y su infraestructura, además del mantenimiento donde muchos prefieren utilizar métodos de prevención y de control artesanales. Por ende, muchos productores dependen de sus propios medios para garantizar un cacao de calidad libre de enfermedades.

Los recientes estudios sobre aprendizaje profundo (deep learning) han logrado grandes avances gracias a los algoritmos representados por redes neuronales convolucionales (CNN), logrando así mejorar características como la precisión y la velocidad de detección a comparación de las técnicas tradicionales.

En el ámbito del reconocimiento de imágenes con deep learning, la detección objetos ha permitido abordar varios problemas que incluyen un escenario y varios objetos a detectar. En consecuencia, la principal diferencia entre los algoritmos de detección de objetos y los algoritmos de clasificación es que los algoritmos de detección pretenden ubicar el objeto de interés y dibujar un cuadro delimitador.

Dentro de la detección de objetos se han integrado dos tipos principales de algoritmos: algoritmos de una etapa y algoritmos de dos etapas. Según [1] la principal diferencia entre ambos es la velocidad con la que se genera un resultado, al realizar en dos etapas la detección y clasificación se conduce a una velocidad más lenta. Por lo tanto, los algoritmos de una etapa pueden generar un cuadro delimitador y la categoría directamente.

YOLO es un algoritmo perteneciente a la familia de detección de objetos de una etapa, en la actualidad se lo considera como uno de los algoritmos más famosos dentro de los detectores gracias a sus constantes innovaciones desde su primer lanzamiento en 2016. [2] Este algoritmo se ha establecido como tendencia para

abordar la detección y el seguimiento de vehículos o personas por su velocidad de inferencia y practicidad.

Considerando lo anterior, el presente trabajo tiene como finalidad desarrollar un modelo para la detección de objetos entrenado con YOLOv5 para la detección de mazorcas de cacao enfermas que será integrado a una aplicación móvil. Para su desarrollo se consideró la recopilación de imágenes de mazorcas saludables y afectadas con fitóftora y monilia. Además de la preparación del dataset utilizando LabelImg y Roboflow.

Por otro lado, para el desarrollo del prototipo se entrenó el modelo utilizando la versión pequeña de YOLOv5 y la ejecución de las librerías de las tecnologías ONNX y Tencent NCNN para la obtención de un modelo apto para el ámbito móvil empleando la metodología Mobile-D. El desarrollo de la aplicación se llevó a cabo en el IDE Android Studio.

El documento está organizado por Capítulo 1: diagnóstico de necesidades y requerimientos; donde se contextualiza la importancia del cacao en el Ecuador y la necesidad de un sistema para detectar sus enfermedades mediante deep learning. Capítulo 2: desarrollo del prototipo; en esta sección detallan los fundamentos teóricos, planteamiento de objetivos y los procesos para llevar a cabo el modelo junto a la aplicación móvil. Capítulo 3: evaluación de prototipo; se aplica la evaluación del modelo a través de una matriz de confusión y se obtienen resultados sobre las métricas.

CAPÍTULO I: DIAGNÓSTICO DE NECESIDADES Y REQUERIMIENTOS

1.1. Ámbito de Aplicación: descripción del contexto y hechos de interés

El cacao es uno de los principales productos tradicionales de exportación ecuatoriana. Según Instituto Nacional de Estadística y Censos (INEC) el sector cacaotero es una base fundamental para la economía de las familias de la costa, sierra y oriente al contribuir con el 5% de la población económicamente activa nacional (PEA) y el 15% rural. [3]

En promedio, en el país existen 550 hectáreas de producción de cacao. Solo en 2018, la producción alcanzó 315.000 toneladas métricas y se exportaron $ 680 millones. [4]

En base a [5] la producción del cacao puede verse afectada con un 80% de pérdida del cultivo con enfermedades como la monilia y la escoba de bruja. Otras enfermedades conocidas son: el mal de machete, la mazorca negra y la muerte regresiva. Por ende, las enfermedades resultan ser una gran limitante para los productores de cacao que pueden llegar a perder gran parte de su producción si no se hace una adecuada inversión de recursos en el control y manejo de estas.

En la actualidad con los avances tecnológicos, se ha visto la necesidad de buscar soluciones en los sistemas informáticos. Es así, como la Inteligencia Artificial (IA) ha recibido una atención significativa debido a sus capacidades para resolver problemas complejos del mundo real. Con los nuevos enfoques de aprendizaje automático que incluyen redes neuronales artificiales, existen ya sistemas que obtienen un mejor desempeño que los humanos en diferentes dominios. [6]

Deep learning es una especie de algoritmo de aprendizaje automático que trabaja con redes neuronales multicapa. [7] Según [8], los campos en los que ha destacado esta técnica son el reconocimiento de imágenes, la detección de objetos y puntos clave, la estimación de pose, el reconocimiento de voz y la segmentación de objetos.

En los trabajos relacionados a la diagnóstico de enfermedades de cacao, se destaca [9] donde utilizan imágenes de cacaos con la enfermedad fitóftora y sin la enfermedad, y se aborda el problema utilizando deep learning con la red neuronal convolucional ResNet18 para la clasificación de imágenes de

obteniendo como resultado una precisión de predicción del 83% en el entrenamiento como 25 épocas. En [10] se maneja el problema de las enfermedades y pestes en la mazorca mediante la clasificación de imágenes con aprendizaje profundo y el pre procesamiento de imagen, como resultado final se desarrolló una aplicación móvil que arroja una precisión del 84,87% para el reconocimiento de enfermedades y 80,80% para el reconocimiento de pestes.

En vista que los trabajos mencionados han utilizado la técnica de clasificación de imágenes con deep learning, el presente proyecto plantea el uso de deep learning con detección de objetos debido a que permite localizar y reconocer varias instancias de objetos en una sola imagen a diferencia de la primera técnica mencionada.

Por ello, como propuesta para abordar el problema de detección de enfermedades que afectan a la mazorca de cacao (monilia y fitóftora) se hará uso de técnicas de deep learning para la detección de objetos que se integraran en una aplicación móvil como producto final.

1.2. Establecimiento de Requerimientos

YOLO (You Only Look Once) es un modelo de detección de objetos que hace uso de una red neuronal convolucional que predice cuadros delimitadores y la clase para estos cuadros.

Su última versión YOLOv5 desarrollado por el equipo de Ultralytics LLC, proporciona varias ventajas, además de su arquitectura simple resulta más rápido y más robusto que otros miembros de la familia YOLO. Según [11], la versión YOLOv5 obtuvo mejores resultados en el tiempo de entrenamiento, peso del modelo final y rendimiento en la detección que sus versiones anteriores más populares.

Según los autores [12] resalta YOLOv5 en el hecho de presentar cuatro tipos de tamaños como: YOLOv5s (pequeño), YOLOv5m (mediano), YOLOv5l (grande), y YOLOv5x (extra grande). En un enfoque de implementación en dispositivos móviles se recomienda la versión pequeña.

Es por ello que el presente proyecto de una aplicación móvil que detecta las enfermedades de cacao se basa en la utilización de YOLOv5 en su versión pequeña.

Con el fin de desarrollar una aplicación que aproveche al máximo el potencial de la tecnología YOLO se establecieron los siguientes requerimientos:

- El dataset estará constituido de imágenes en alta resolución de mazorcas enfermas y sanas.
- El dataset poseerá alrededor de 1000 imágenes que se dividirán para el entrenamiento, validación y test.
- La utilización del aprendizaje por transferencia con pesos-pre entrenados.
- Uso del entorno Google Colaboratory para el entrenamiento y pruebas
- Uso del entorno de desarrollo Android Studio para el desarrollo de la aplicación.
- La aplicación permitirá detectar imágenes obtenidas de la galería
- La aplicación permitirá detectar imágenes obtenidas por la cámara
- La aplicación permitirá almacenar las imágenes y visualizarlas en una galería

1.3. Justificación del requerimiento a satisfacer

El cacao es una fruta tropical, sus cultivos se encuentran mayormente en el Litoral y Amazonía ecuatoriana, en los cuales se cultivan dos tipos de cacao: el Cacao CCN-51 y el denominado Cacao Nacional.

Las enfermedades son uno de los factores más limitantes para el cultivo de cacao y al igual que otras especies de plantas pueden afectar negativamente a su producción. Según [13] se describe que las enfermedades que afectan directamente a la mazorca del cacao, como la monilia puede causar pérdidas de hasta el 90%, mientras que la enfermedad fitóftora puede llegar a un 30% del cultivo perdido.

Para los agricultores es de vital importancia la prevención y el diagnóstico oportuno de cualquier afectación que pueda perjudicar a su cultivo. Por ello, es muy importante poseer un adecuado sistema de diagnóstico para tomar decisiones apropiadas.

Para determinar si las plantas poseen enfermedades, existen métodos como el llevar una muestra de la parte afectada de la planta para un análisis de laboratorio o recibir la inspección de un ingeniero agrónomo en el sitio de cultivo, aun así, esto puede significar una desventaja por el tiempo y recursos que puede tomar el proceso.

En la actualidad las herramientas impulsadas por la IA se han vuelto más accesibles y viables en muchos sectores, como lo es en el sector agrícola. Es por eso que se han considerado el enfoque de deep learning que proporciona un acercamiento más exacto a como aprende una mente humana. Las técnicas de visión artificial, permiten a una máquina obtener imágenes y tener la capacidad de percibir lo que pasa en el entorno, mediante el procesamiento de la información y la extracción de su significado.

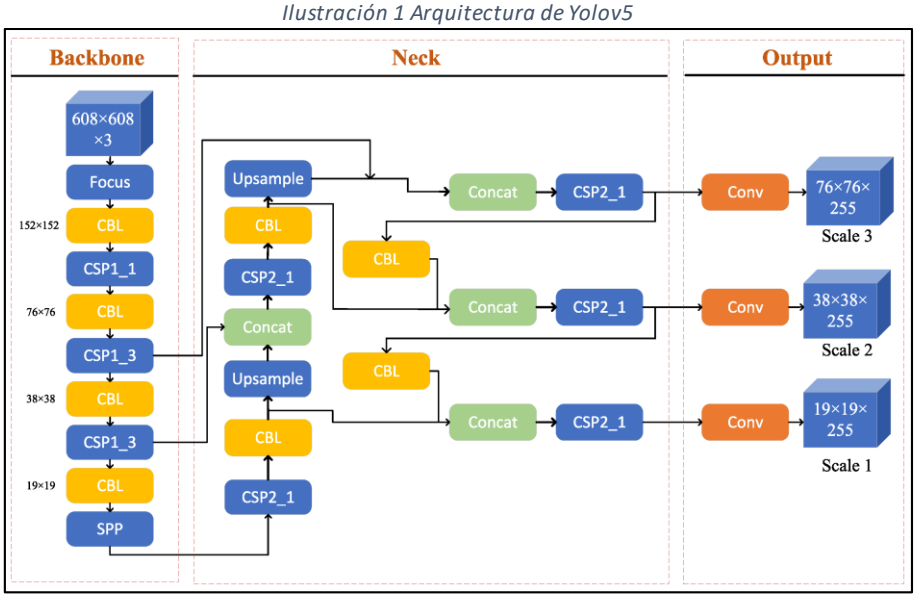
Por lo tanto, en este proyecto integra el entrenamiento de un modelo utilizando el algoritmo de deep learning YOLOv5 y el desarrollo de una aplicación móvil en Android Studio que consumirá dicho modelo permitiendo a los agricultores diagnosticar las afecciones de la mazorca de cacao de forma rápida mediante la captura de imágenes.

2. CAPÍTULO II: DESARROLLO DEL PROTOTIPO

2.1. Definición del prototipo tecnológico

El proyecto propuesto de detección de enfermedades en la mazorca de cacao, se basa en el uso del algoritmo de detección de objetos YOLO. La arquitectura de su versión más reciente se divide en tres secciones denominadas: columna vertebral, cuello y salida como se muestra en la siguiente figura.

Ilustración 1 Arquitectura de Yolov5

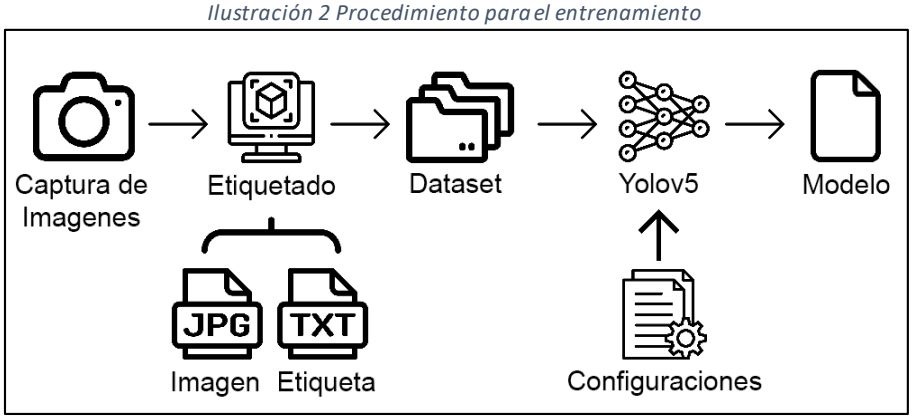


Fuente: [14]

La primera sección tiene el objetivo de extraer las características en las imágenes, siguiendo con el cuello, se utiliza la información extraída y la fusiona para generar tres escalas de mapas de entidades, finalmente la salida detecta los objetos de estos mapas generados.

El entrenamiento de un modelo con el algoritmo YOLOv5, parte de la construcción del conjunto de imágenes etiquetadas con las clases correspondientes, luego debemos preparar un entorno para trabajar, seleccionar nuestro conjunto de imágenes y realizar el entrenamiento con las configuraciones deseadas, al finalizar automáticamente tendremos exportado nuestro modelo.

Ilustración 2 Procedimiento para el entrenamiento

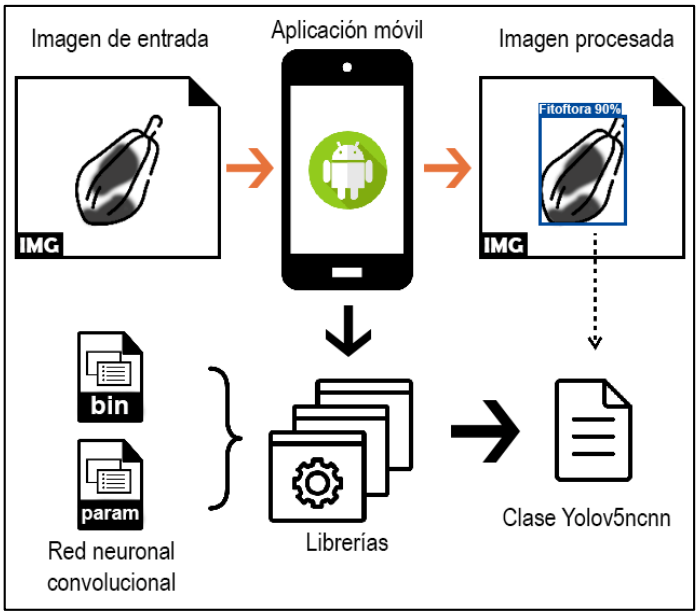


Fuente: Elaboración propia

Para el desarrollo de nuestra la aplicación se hará uso del framework Tencent NCNN que permite la compatibilidad de diversos modelos de redes neuronales en dispositivos móviles. Por ello se debe convertir nuestro modelo al formato admitido por las librerías del framework, obteniendo nuestra red neuronal convolucional en dos archivos de tipo binario y de parámetros.

La aplicación trabajada en Android Studio realizará la detección de las mazorcas enfermas de imágenes tomadas por la cámara o seleccionadas de la galería del dispositivo. Al ejecutarse la detección, el framework procesa la imagen utilizando nuestra red neuronal convolucional y generará las detecciones, la salida dibujara un cuadro delimitador por cada detección junto a su clase con el porcentaje de confianza.

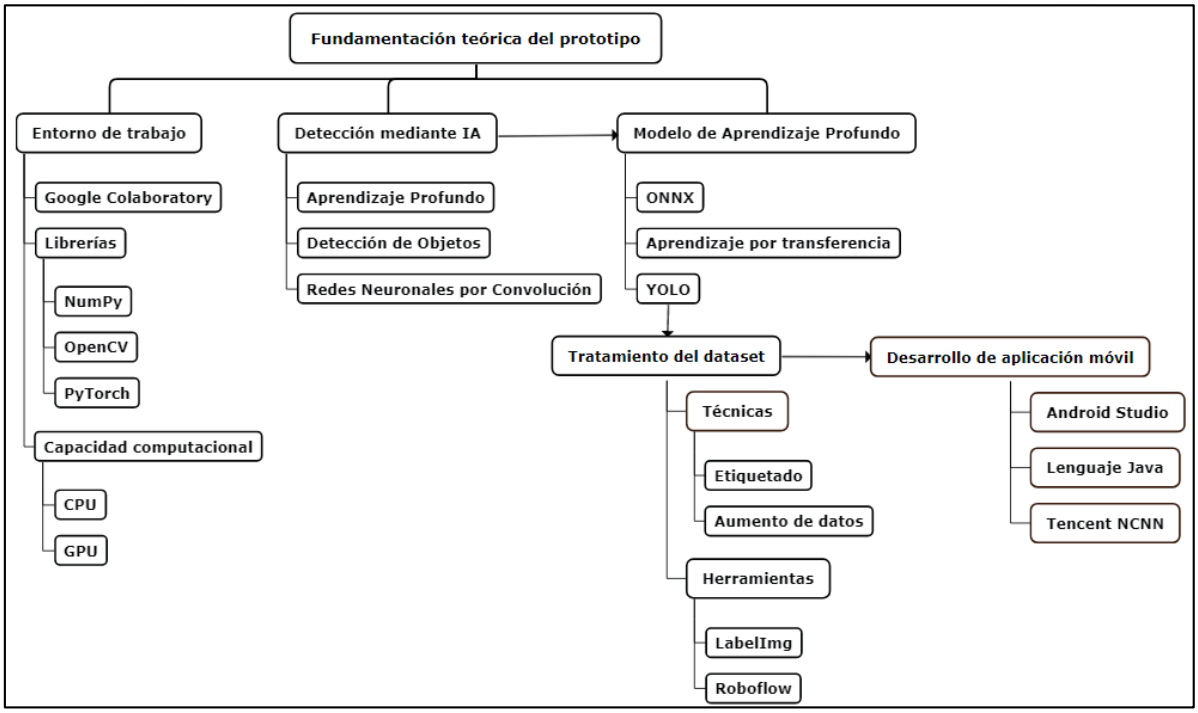
Ilustración 3 Funcionamiento de la aplicación



Fuente: Elaboración propia

2.2. Fundamentación teórica del prototipo

Ilustración 4 Mapa mental de la fundamentación teórica del prototipo



Fuente: Elaboración propia

2.2.1. Detección mediante IA

En la actualidad la inteligencia artificial ha ganado popularidad, por lo que muchos problemas se han encaminado en esta área para la búsqueda de soluciones. Según [15], se segmenta a la IA en varios componentes como: representación del conocimiento, razonamiento, procesamiento del lenguaje natural, aprendizaje automático, visión por computadora y robótica

Debido a los logros de los trabajos en los últimos años, se ha visto a esta técnica en una amplia gama de aplicaciones como la clasificación de imágenes, los automóviles autónomos, el procesamiento del lenguaje natural, el reconocimiento de voz y la atención médica inteligente [16], provocando un efecto transformador en una variedad de industrias.

2.2.1.1. Deep Learning

El deep learning es una especie de algoritmo de aprendizaje automático que trabaja con redes neuronales multicapa. Estos modelos adquieren en su mayoría un pre entrenamiento no supervisado capa por capa y luego utilizan una forma supervisada para ajustar los parámetros. [7]

Algunos ejemplos de tipos de modelos son: red neuronal convolucional (CNN), red de creencias profundas (DBN), codificador automático profundo (DAE), máquina restringida de Boltzmann, entre otros.

Según [8], los campos en los que ha destacado esta técnica son el reconocimiento de imágenes, la detección de objetos y puntos clave, la estimación de pose, el reconocimiento de voz y la segmentación de objetos.

2.2.1.2. Detección de Objetos

La detección de objetos es un enfoque de la IA que utiliza el deep learning para procesar de forma automática grandes volúmenes de datos en formato de imágenes o videos y extraer información de determinados objetos.

Esta técnica localiza objetos en dos fases localización y clasificación, al tener una imagen de entrada ejecuta la detección prediciendo donde se encuentra el objeto y dibuja un cuadro delimitador en la localización, luego en la clasificación reconoce el tipo y le asigna una clase [17], esto se realiza por cada uno de los objetos detectados en la imagen.

En este ámbito los modelos de CNN han demostrado ser eficaces al obtener información de alto nivel de los objetos en la capa de convolución de nivel posterior. [18]

Ilustración 5 Detección de objetos basado en Yolo

Fuente: Elaboración propia

2.2.1.3. Redes Neuronales Convolucionales

Las CNN se han establecido como una herramienta óptima para el desarrollo de soluciones a problemas de reconocimiento de patrones. Su arquitectura [19] se basa en el mecanismo de percepción visual natural de los seres vivos y esta consta principalmente de capas como: capas convolucionales, capas de agrupación y capas fully-connected.

El hecho de trabajar una red neuronal con múltiples capas ha hecho posible obtener mejores representaciones de característica permitiendo reconocer patrones visuales directamente a partir de píxeles con poco o ningún pre procesamiento. Por ello se han utilizado con éxito en los campos de la visión por computadora, como el reconocimiento de imágenes, la detección de objetos y la segmentación semántica. [20]

2.2.2. Entorno de trabajo

Las tareas de deep learning están programadas para ser ejecutadas en GPU debido a los cálculos pesados que son necesarios para aprender un conjunto de datos. Actualmente existen soluciones en la nube, servicios que ofrecen el acceso gratuito a hardware totalmente configurado para fines educativos y de investigación.

Para este proyecto el entorno de trabajo engloba el servicio en la nube Google Colaboratory y las distintas librerías que se instalaron para realizar el entrenamiento del modelo.

2.2.2.1. Google Colaboratory

Colaboratory es un proyecto de Google que ofrece el uso gratuito a una GPU que se ejecutan en la nube para ayudar a desarrolladores a trabajar en un entorno Jupyter portátil donde pueden escribir, editar y ejecutar código en Python. [21]

Google Colaboratory es una opción simple y rentable para trabajar en un escenario con recursos basados en GPU sin recurrir a la compra de estos. Colaboratory ofrece gratuitamente la GPU Tesla K80, que se puede acceder al

activar la opción de GPU en el libro de trabajo, y puede resultar 45 veces más rápida que una CPU convencional. [22]

2.2.2.2. Librerías

YOLOv5 depende de varias librerías para su correcto funcionamiento por lo que es un aspecto técnico importante de preparar al momento de ejecutar el entrenamiento. Las librerías son importantes ya que permiten añadir funciones a una aplicación para que esta haga uso de estas.

2.2.2.2.1. NumPy

Numpy es un paquete de Python que proporciona una variedad de operaciones matemáticas y lógicas con matrices. Según [23] NumPy añade funciones que permiten realizar operaciones matemáticas en matrices como rutinas estadísticas, algebraicas y trigonométricas.

2.2.2.2.2. OpenCV

OpenCV es una librería de código abierto que surgió en Intel en 1999, es ampliamente utilizada en proyectos de visión artificial por lo que es una de las más populares en aplicaciones de este tipo. Esta librería es muy potente y hace uso de otras para facilitar sus operaciones como NumPy. [24]

2.2.2.2.3. PyTorch

PyTorch es una librería escrita en Python de alto uso en implementaciones de deep learning, se basa en el método “Define-by-Run” que se traduce a definir mientras opera. Esta librería resalta por su forma de trabajar al optimizar el uso de la memoria de la GPU y CPU. [25]

2.2.2.3. Capacidad computacional

El rendimiento de los modelos deep learning han adquirido tan buenos resultados por dos razones: la cantidad de datos disponibles en la red para el entrenamiento y el poder computacional de los procesadores de última generación. Esta última es debido a numerosos y complejos cálculos que exigen de soluciones de hardware efectivas.[26]

2.2.2.3.1. CPU

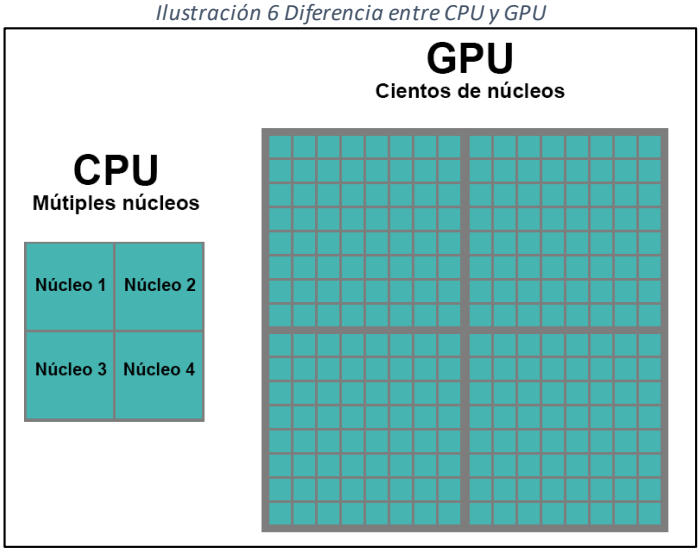
La unidad central de procesamiento (CPU) es el procesador convencional que utilizan los computadores, este suele tener menos procesadores que los GPU por lo que su rendimiento para tareas de deep learning es menos eficiente. [27]

2.2.2.3.2. GPU

Las unidades de procesamiento gráfico (GPU) son consideradas como una de las tecnologías bases para las tareas de aprendizaje automático, dado el crecimiento de escala de las redes neuronales profundas actuales, los procesos que conllevan su formación requieren mucha carga de trabajo por lo que la arquitectura paralela y la gran cantidad de núcleos de las GPU ha resultado como una opción óptima. [28]

La aceleración de GPU demuestra mayor rendimiento gracias a las ventajas que posee sobre las CPU, según [29] su gran ancho de banda de memoria permite cumplir con los procesos de entrenamiento en tiempos más reducidos. Además, la red neuronal puede beneficiarse del paralelismo de las GPU al permitir que cada neurona se procese independientemente.

Ilustración 6 Diferencia entre CPU y GPU



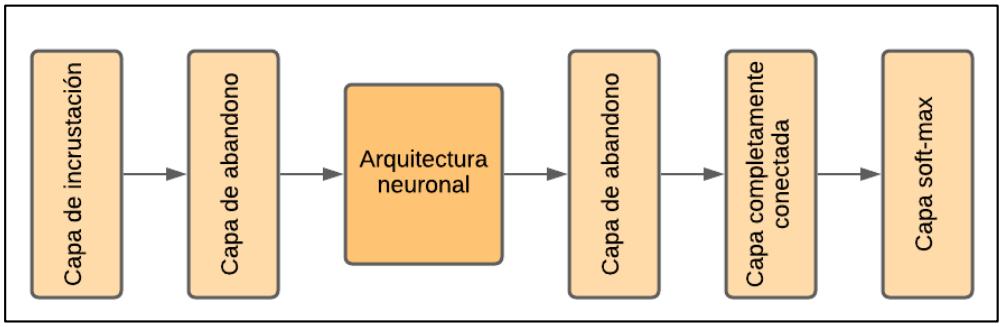
Fuente: Elaboración propia

2.2.3. Modelo de Deep learning

Los modelos de deep learning generalmente son arquitecturas que basadas en varios millones de parámetros que generan matrices dentro de sus cálculos. En

base al trabajo [30] este tipo de modelos se componen de los siguientes elementos: capa de incrustación, arquitectura neuronal, capa de abandono, capa completamente conectada y como se muestra en la siguiente figura.

Ilustración 7 Arquitectura de los modelos de deep learning



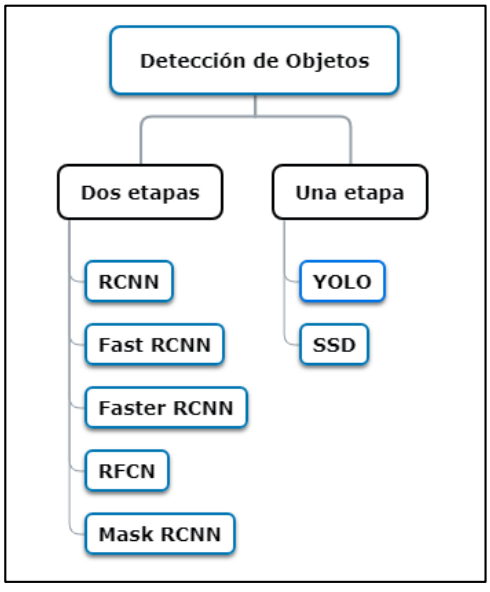
Fuente: Elaboración propia

2.2.3.1. YOLO

YOLO es una familia de modelos de detección de objetos, presenta una innovación al convertir el problema de detección de objetos en un problema de regresión. A partir de la imagen de entrada, devuelve directamente el cuadro delimitador de salida y sus categorías de clasificación en múltiples objetivos. [31]

La idea de utilizar la regresión vuelve más sencillo el aprendizaje de las características del objetivo y optimiza la velocidad de detección. La familia YOLO utiliza una red neuronal de una etapa para lograr el posicionamiento y clasificación del objeto detectado sin dividirlo en dos fases. [32]

Ilustración 8 Algoritmos de una y dos etapas



Fuente: Elaboración propia

En la actualidad se ha desarrollado hasta su quinta versión (YOLOv5) y posee un potencial de crecimiento por su comunidad activa, además en comparación a otros algoritmos de deep learning populares en la detección de objetos como SSD y Faster-RCNN, ha presentado un mejor rendimiento general. [33]

2.2.3.2. ONNX

Intercambio de redes neuronales abiertas (ONNX) es un formato de archivo desarrollado por varias empresas como Microsoft, Amazon, Facebook e IBM, su finalidad es el de almacenar modelos entrenados de deep learning utilizando el búfer de protocolo de Google (protobuf). [34]

Según [35] ONNX facilita a los desarrolladores la creación de modelos de IA con un modelo gráfico extensible, definiciones de operadores integrados y tipos de datos estándar para su implementación para inferencia.

2.2.3.3. Aprendizaje por transferencia

Una tarea muy difícil en el desarrollo de modelos de deep learning es conseguir una cantidad suficiente de imágenes a gran escala, por ello para resolver este problema se aplica la técnica de aprendizaje por transferencia.[36]

El aprendizaje por transferencia es una característica de las redes neuronales convolucionales para transferir el aprendizaje de un problema de origen a un problema de destino. Según [37] transferir el aprendizaje significa usar un modelo entrenado previamente de mayor tamaño en el entrenamiento de un nuevo modelo que podría tener un tamaño menor de datos, a través de esto el nuevo modelo aprende características como bordes, colores, formas, etc. Además, esta técnica permite optimizar el entrenamiento obteniendo reduciendo el tiempo del entrenamiento y maximizando el rendimiento. [38]

2.2.4. Tratamiento del dataset

Un conjunto de datos (dataset) cuenta con las imágenes que se usan para el aprendizaje de una red neuronal, pero es importante que cumpla con cierto formato dependiendo del algoritmo a entrenar. Para esta tarea se llegan a utilizar diferentes técnicas y herramientas que permiten la preparación del conjunto de datos para el entrenamiento.

2.2.4.1. Técnicas

2.2.4.1.1. Etiquetado

El etiquetado de imágenes es una tarea que se debe realizar en algunos proyectos de aprendizaje automático para especificar la posición del objeto en la imagen mediante una etiqueta. Con la ayuda de una herramienta se debe encerrar el objeto en una forma rectangular y asignar una etiqueta como, por ejemplo: perro, gato, etc.

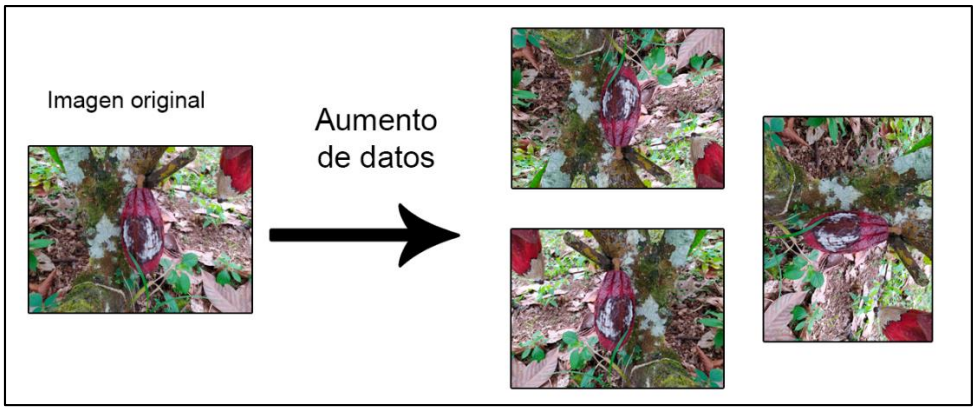
Según [39] las etiquetas en YOLO son un archivo en formato txt y poseen los valores de las coordenadas de la esquina superior izquierda y la esquina inferior derecha que conforman el cuadro delimitador. Además de un índice que hace referencia a la clase.

2.2.4.1.2. Aumento de datos

La técnica de aumento de datos en inglés data augmentation, permite generar una diversidad de imágenes a partir del conjunto de datos original para así aumentar la cantidad de ejemplares en una o varias clases en particular. En base a [11] el uso del aumento de datos mejora el rendimiento de la red neuronal profunda durante el entrenamiento del modelo, evitando así el sobreajuste.

Esta técnica ayuda a aumentar la cantidad de información que se puede extraer del conjunto de datos de forma artificial, algunos métodos usados son: transformaciones geométricas y de color, borrado aleatorio, entrenamiento de adversarios y transferencia de estilo neuronal. [40]

Ilustración 9 Obtención del aumento de datos



Fuente: Elaboración propia

2.2.4.2. Herramientas

2.2.4.3. LabelImg

LabelImg es una popular herramienta gráfica de código abierto desarrollado por Tzuta Lin para el etiquetado de imágenes, está escrita en Python y utiliza el framework Qt para su interfaz gráfica. Las anotaciones son compatibles con formatos PASCAL VOC, YOLO y CreateML. [41]

2.2.4.4. Roboflow

Roboflow es una plataforma de visión por computadora que permite a los desarrolladores subir sus conjuntos de datos y a su vez explorar una colección de datos públicos. Tiene una potente variedad de utilidades para trabajar datos personalizados como por ejemplo dibujar anotaciones, orientación de imagen, cambio de tamaño, contraste y aumento de datos. Además, se puede entrenar modelos. [42]

2.2.5. Desarrollo de aplicación móvil

Para diseñar una aplicación móvil se necesita de tecnología capaz de cumplir con los objetivos planteados. En el aspecto técnico de desarrollo se optado por la herramienta Android Studio ya que posee muchas funciones para generar un producto de alta calidad.

2.2.5.1. Android Studio

Android Studio es un entorno de desarrollo integrado (IDE) para el desarrollo de aplicaciones Android disponible para plataformas de escritorio Mac, Windows y Linux, posee un potente editor de código, compilador y emulador entre otras funciones. [43]

Android posee un modelo de programación que consiste en que las aplicaciones se organizan como actividades, una actividad equivale a una pantalla individual en la que un usuario puede interactuar. [44]

2.2.5.2. Lenguaje Java

Es un lenguaje de programación orientado a objetos lanzado a mediados de la década de los 90 por Sun Microsystems y se considera como una tecnología

base para muchos servicios y aplicaciones. Actualmente se sigue innovando en su implementación y en el desarrollo móvil tiene un papel muy importante siendo uno de los lenguajes más utilizados en aplicaciones Android. [45]

2.2.5.3. Tencent NCNN

NCNN es un framework de inferencia de redes neuronales de alto rendimiento optimizado para plataformas móviles desarrollado por la compañía china de Internet Tencent. Este marco no posee dependencias de terceros, es multiplataforma y se ejecuta más rápido que todos los marcos de código abierto conocidos en la CPU del teléfono móvil. [46]

Permite ejecutar varios modelos de deep learning como YOLO mediante su propio formato de modelo, para ello es necesario utilizar las herramientas proporcionados en su repositorio de Github.

2.3. Objetivos del prototipo

2.3.1. Objetivo General

- Desarrollar una aplicación móvil que permita la detección de enfermedades en las mazorcas de cacao mediante la aplicación de técnicas de deep learning.

2.3.2. Objetivos Específicos

- Caracterizar el estado del arte de la aplicación de modelos de deep learning.
- Definir los procesos para la captura y etiquetado de imágenes para generar un conjunto de datos.
- Evaluar el modelo entrenado en la red neuronal convolucional proporcionada por el algoritmo YOLOv5.
- Diseñar un prototipo de aplicación móvil para la detección de enfermedades en la mazorca de cacao.

2.4. Diseño del prototipo

El diseño del prototipo consta de dos fases, la primera consta de la generación del conjunto de datos y luego la ejecución del entrenamiento de la red neuronal convolucional de YOLOv5.

2.4.1. Generación del conjunto de datos

2.4.1.1. Recolección y captura de imágenes

Para generar el conjunto de datos de las enfermedades de cacao que afectan a la mazorca como lo son la Fitóftora y Monilia se consideró el conjunto de datos "Enfermedades de Cacao" disponible en Kaggle [47] que consta de dichas clases, además de la clase Sana, que hace referencia a mazorcas sin ninguna enfermedad.

Además, se realizó la captura de imágenes de mazorcas enfermas y sanas en una finca de la parte alta de Cantón Santa Rosa, en su mayoría de tipo CCN-51, y pocos ejemplares del cacao nacional. En total se seleccionaron 680 imágenes para formar parte del conjunto de datos final.

Las imágenes se tomaron con un dispositivo Redmi Note 8, que consta de una cámara frontal de 48 megapixeles, dando como resultado imágenes de alta calidad. En la toma de fotos no se consideró ningún aspecto técnico para la toma de fotos, estas se tomaron en diferentes ángulos y condiciones de iluminación y solo se consideró el correcto enfoque para la mayor nitidez.

2.4.1.2. Etiquetado del conjunto de datos

El etiquetado de las imágenes se realizó para las imágenes capturadas, para esta tarea se utilizó la herramienta LabelImg y se procedió a dibujar los rectángulos en las mazorcas y asignar etiquetas correspondientes a las clases en cada una de las imágenes.

Ilustración 10 Herramienta de etiquetado de imágenes

Fuente: Elaboración propia

2.4.1.3. Aumento del conjunto de datos

Debido a que la cantidad de imágenes capturadas para cada clase fueron disparejas se decidió aplicar el aumento de datos mediante transformación en las clases fifoftora y monilia. Esto se realizó utilizando la librería de clasificación de imágenes CLoDSA buscando de esta manera obtener nuevas imágenes.

Para ello se utilizó un cuaderno en Google Colaboratory diseñado específicamente para esta tarea, y se realizaron los siguientes pasos:

1. Se instalan las librerías y se importan las dependencias.

Ilustración 11 Instalación de dependencias para el aumento de datos

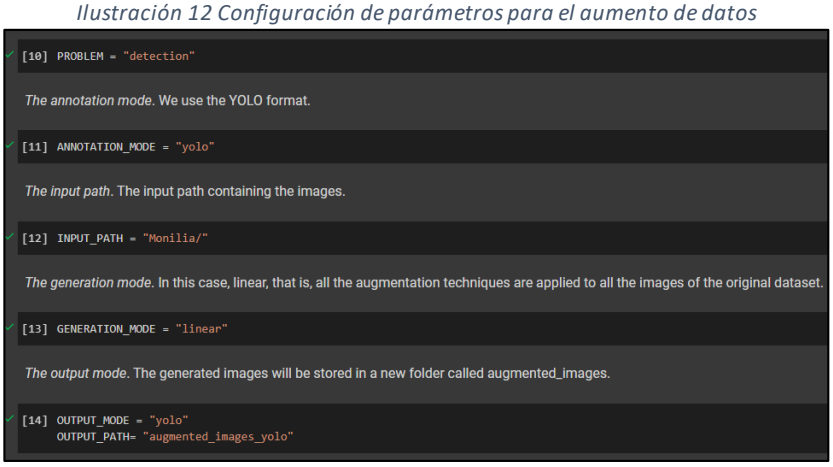
```
!pip install clodsa

Collecting clodsa
  Downloading clodsa-1.2.47.tar.gz (30 kB)
Collecting mahotas
  Downloading mahotas-1.4.12-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (5.7 MB)
     |████████████████████████████████| 5.7 MB 4.8 MB/s
Requirement already satisfied: imutils in /usr/local/lib/python3.7/dist-packages (from clodsa) (0.5.4)
Requirement already satisfied: Keras in /usr/local/lib/python3.7/dist-packages (from clodsa) (2.7.0)
Collecting commentjson
  Downloading commentjson-0.9.0.tar.gz (8.7 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from clodsa) (1.4.1)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from clodsa) (3.1.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from clodsa) (1.19.5)
Requirement already satisfied: progressbar2 in /usr/local/lib/python3.7/dist-packages (from clodsa) (3.38.0)
Requirement already satisfied: scikit_learn in /usr/local/lib/python3.7/dist-packages (from clodsa) (1.0.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from clodsa) (1.1.0)
Collecting lark-parser<0.8.0,>=0.7.1
  Downloading lark-parser-0.7.8.tar.gz (276 kB)
     |████████████████████████████████| 276 kB 24.8 MB/s
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py->clodsa) (1.5.2)
Requirement already satisfied: python-utils>=2.3.0 in /usr/local/lib/python3.7/dist-packages (from progressbar2->clodsa) (2.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from progressbar2->clodsa) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit_learn->clodsa) (3.0.0)
Building wheels for collected packages: clodsa, commentjson, lark-parser
  Building wheel for clodsa (setup.py) ... done
  Created wheel for clodsa: filename=clodsa-1.2.47-py2.py3-none-any.whl size=74311 sha256=5588cfd052d6972fa12a757ee2b8675e1a5a39f0bf92dc1ee1db5d141fcca6c1
```

Fuente: Elaboración propia

2. Se especifican los parámetros necesarios para definir el tipo de datos

Ilustración 12 Configuración de parámetros para el aumento de datos



Fuente: Elaboración propia

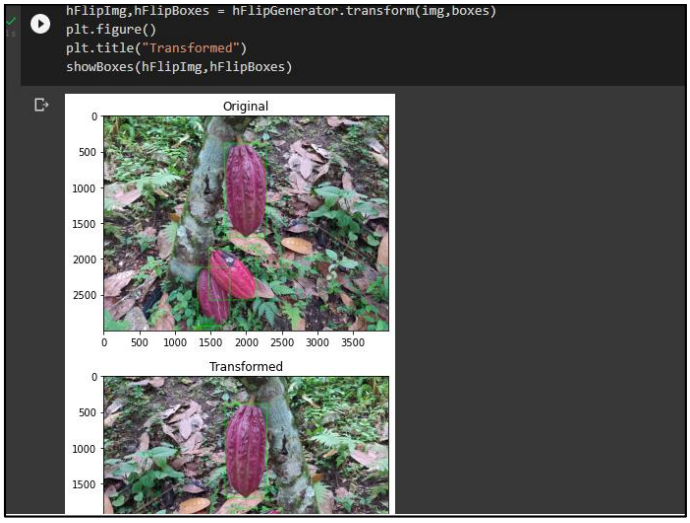
3. Definimos la función para leer las etiquetas y las visualizamos

Ilustración 13 Visualización de etiquetas para el aumento de datos

Fuente: Elaboración propia

4. Finalmente aplicamos las transformaciones y generamos los nuevos datos

Ilustración 14 Generación del aumento de datos



Fuente: Elaboración propia

Después de este proceso se agregaron las nuevas imágenes al conjunto de datos final. La formación del conjunto de datos se detalla en la siguiente tabla.

Tabla 1 Resumen de la generación del dataset

Clase	Datos tomados de Kaggle	Datos propios	Aumento de datos	Total
Fitóftora	107	279	60	446
Monilia	105	78	234	417
Sana	100	323	-	423
Total				1286

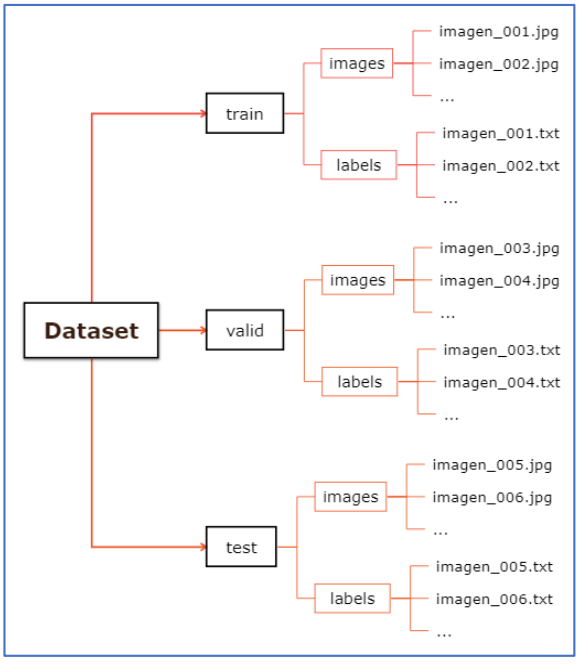
Fuente: Elaboración propia

Dado que el presente proyecto es del ámbito de detección de objetos, se debe especificar que existen casos en que las imágenes tendrán varias etiquetas de objetos y es una variable que representa una mayor cantidad de mazorcas que imágenes totales.

2.4.1.4. División del conjunto de datos

Para el empleo del conjunto de datos se debe organizar en tres conjuntos denominados: train (entrenamiento), valid (validación) y test (prueba). Cada una de estas carpetas deben poseer cierta cantidad de imágenes con sus etiquetas como se muestra en la siguiente estructura.

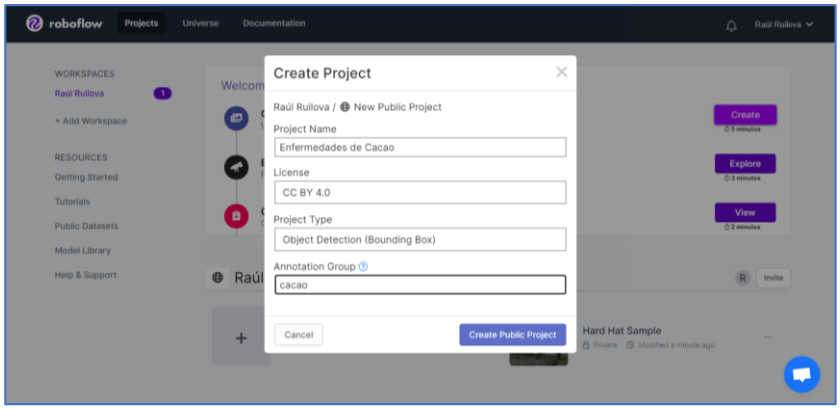
Ilustración 15 Estructura del dataset



Fuente: Elaboración propia

Por ende, se usó la plataforma Roboflow donde se creó un proyecto denominado Enfermedades de Cacao y se siguió los siguientes pasos.

Ilustración 16 Creación del proyecto en Roboflow



Fuente: Elaboración propia

1. Subir cada una de las imágenes junto a los archivos txt que corresponden a las etiquetas, además de un archivo denominado “classes” que detalla el nombre de las clases.

Ilustración 17 Carga de los datos a Roboflow

Fuente: Elaboración propia

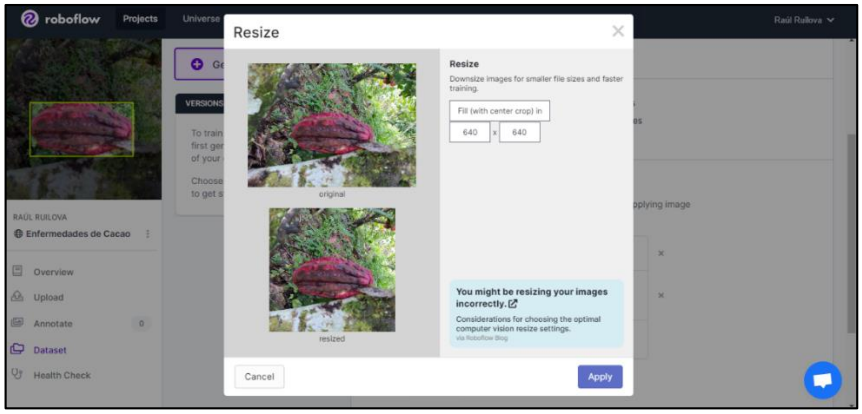
2. Cuando haya finalizado la subida, se debe seleccionar cómo se van a dividir las imágenes, en este caso se utilizó la configuración por defecto con 70% para entrenamiento, 20% para validación y 10% para pruebas.

Ilustración 18 División del dataset en Roboflow

Fuente: Elaboración propia

3. Luego se pueden aplicar ciertos ajustes a los datos, se configuro la redimensión de las imágenes a 640x640 manteniendo la relación de aspecto.

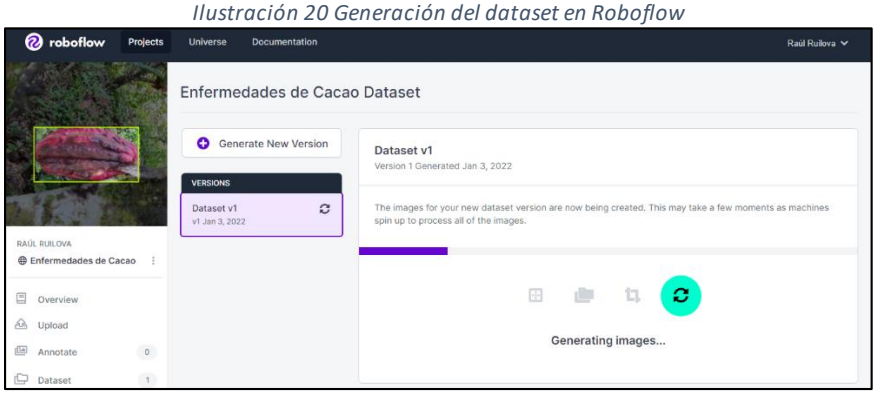
Ilustración 19 Configuración de ajustes al dataset en Roboflow



Fuente: Elaboración propia

4. A continuación, se genera la versión con los ajustes aplicados y se asigna un nombre.

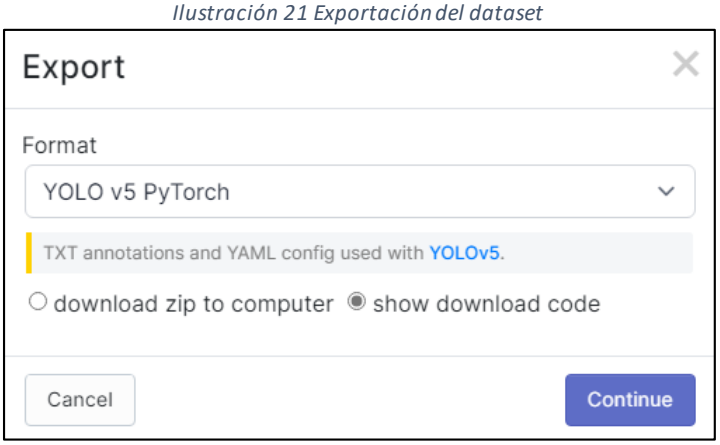
Ilustración 20 Generación del dataset en Roboflow



Fuente: Elaboración propia

5. Ahora debemos exportar la versión de nuestro conjunto de datos, para ello seleccionamos el formato que necesitamos, en nuestro caso YOLOv5 Pytorch.

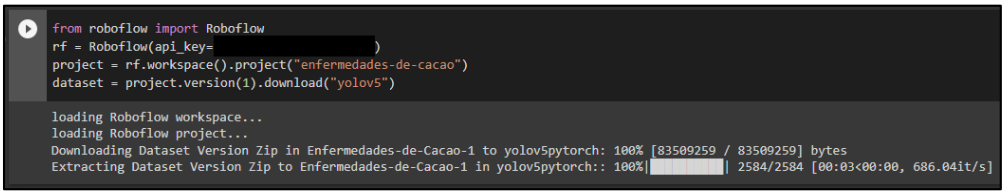
Ilustración 21 Exportación del dataset



Fuente: Elaboración propia

6. Utilizamos el código de Jupyter que nos facilita la plataforma para la exportación y lo ejecutamos en un cuaderno de Google Colaboratory, de esta forma se procederá a descargar nuestro conjunto de datos dentro de nuestra nube para utilizarlo.

Ilustración 22 Descarga del dataset generado

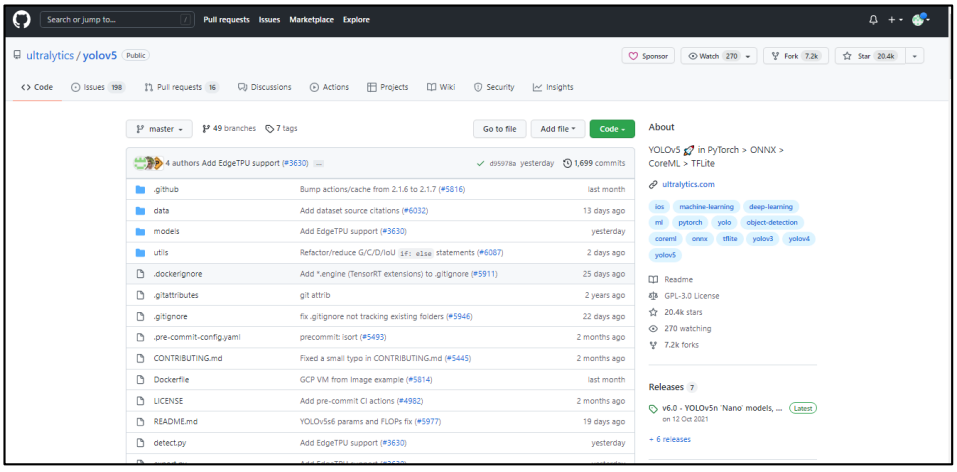


Fuente: Elaboración propia

2.4.2. Entrenamiento del modelo

Para el entrenamiento de nuestro modelo se siguió la documentación del repositorio oficial de YOLOv5. Los archivos se clonaron desde Github a nuestra nube y se realizaron los siguientes pasos.

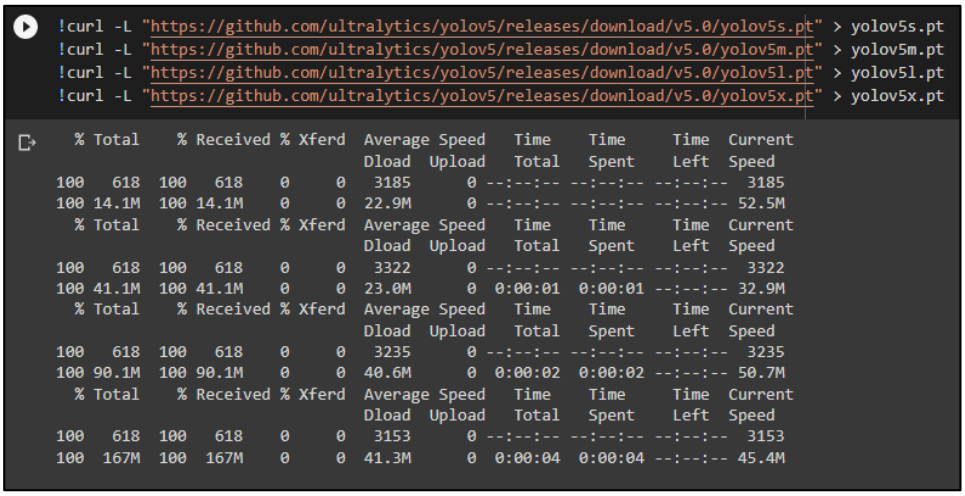
Ilustración 23 Repositorio oficial de YOLOv5 en Github



Fuente: Elaboración propia

1. Se descarga los pesos pre-entrenados desde el repositorio utilizando los siguientes comandos.

Ilustración 24 Descargar de los presos pre-entrenados



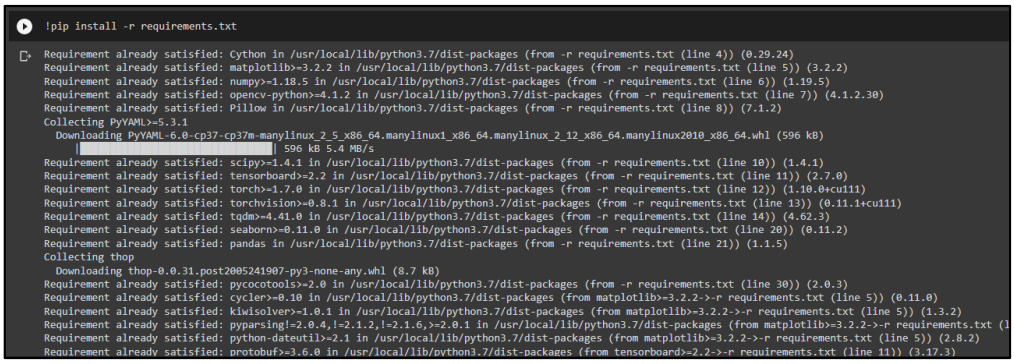
```
!curl -L "https://github.com/ultralytics/yolov5/releases/download/v5.0/yolov5s.pt" > yolov5s.pt
!curl -L "https://github.com/ultralytics/yolov5/releases/download/v5.0/yolov5m.pt" > yolov5m.pt
!curl -L "https://github.com/ultralytics/yolov5/releases/download/v5.0/yolov5l.pt" > yolov5l.pt
!curl -L "https://github.com/ultralytics/yolov5/releases/download/v5.0/yolov5x.pt" > yolov5x.pt
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   618  100   618    0     0   3185      0 --:--:-- --:--:-- --:--:--  3185
100 14.1M  100 14.1M    0     0  22.9M      0 --:--:-- --:--:-- --:--:-- 52.5M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   618  100   618    0     0   3322      0 --:--:-- --:--:-- --:--:--  3322
100 41.1M  100 41.1M    0     0  23.0M      0  0:00:01  0:00:01 --:--:-- 32.9M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   618  100   618    0     0   3235      0 --:--:-- --:--:-- --:--:--  3235
100 90.1M  100 90.1M    0     0  40.6M      0  0:00:02  0:00:02 --:--:-- 50.7M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   618  100   618    0     0   3153      0 --:--:-- --:--:-- --:--:--  3153
100  167M  100  167M    0     0  41.3M      0  0:00:04  0:00:04 --:--:-- 45.4M
```

Fuente: Elaboración propia

2. Se instalan los requerimientos desde el archivo de texto que posee todos comandos de las dependencias y librerías necesarias para la funcionalidad de YOLOv5.

Ilustración 25 Instalación de los requerimientos de YOLOv5



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

```
Requirement already satisfied: Cython in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 4)) (0.29.24)
Requirement already satisfied: matplotlib>=3.2.2 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 5)) (3.2.2)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 6)) (1.19.5)
Requirement already satisfied: opencv-python>=4.1.2 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 7)) (4.1.2.30)
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 8)) (7.1.2)
Collecting PyYAML>=5.3.1
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (596 kB)
     |████████████████████████████████| 596 kB 5.4 MB/s
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 10)) (1.4.1)
Requirement already satisfied: tensorboard>=2.2 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 11)) (2.7.0)
Requirement already satisfied: torch>=1.7.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 12)) (1.10.0+cu111)
Requirement already satisfied: torchvision>=0.8.1 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 13)) (0.11.1+cu111)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 14)) (4.62.3)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 20)) (0.11.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 21)) (1.1.5)
Collecting thop
  Downloading thop-0.0.31.post2005241907-py3-none-any.whl (8.7 kB)
Requirement already satisfied: pycocotools>=2.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 30)) (2.0.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (1.3.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (l
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (2.8.2)
Requirement already satisfied: protobuf>=3.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (3.17.3)
```

Fuente: Elaboración propia

En la siguiente tabla se detalla los paquetes y la versión correspondiente instalación.

Tabla 2 Resumen de requerimientos de YOLOv5

Nombre del paquete	Versión
Cython	-
matplotlib	>=3.2.2
numpy	>=1.18.5
opencv-python	>=4.1.2
Pillow	-
PyYAML	>=5.3.1
scipy	>=1.4.1
tensorboard	>=2.2
torch	>=1.7.0
torchvision	>=0.8.1
tqdm	>=4.41.0

Fuente: Elaboración propia

3. Se ejecuta el entrenamiento proporcionando los parámetros para entrenar el modelo con nuestro conjunto de datos con el archivo train.py.

Ilustración 26 Ejecución del entrenamiento

```
!python train.py --img 640 --batch 16 --epochs 100 --data Enfermedades-de-Cacao-1/data.yaml --weights weights/yolov5s.pt

github: skipping check (not a git repository)
YOLOv5  torch 1.10.0+cu111 CUDA:0 (Tesla K80, 11441.1875MB)

Namespace(adam=False, batch_size=16, bucket='', cache_images=False, cfg='', data='Enfermedades-de-Cacao-1/data.yaml', device='', epochs=100,
wandb: Install Weights & Biases for YOLOv5 logging with 'pip install wandb' (recommended)
Start Tensorboard with "tensorboard --logdir runs/train", view at http://localhost:6006/
hyperparameters: lr0=0.01, lrf=0.2, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.0
Overriding model.yaml nc=80 with nc=3
```

Fuente: Elaboración propia

Los parámetros completos utilizados para el entrenamiento incluyen el tamaño de la imagen, el valor de épocas y tamaño de lote, la dirección de la data y de los pesos pre-entrenados. Los valores finales se detallan en la tabla 3.

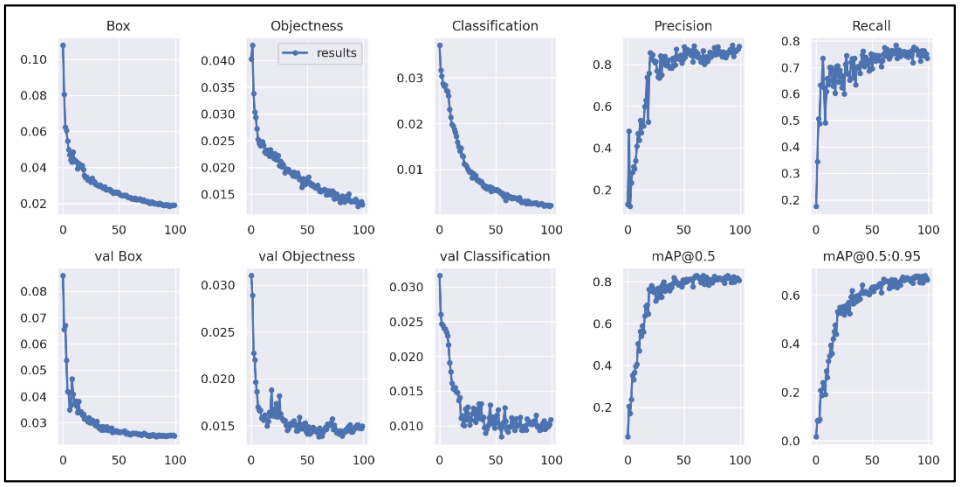
Tabla 3 Parámetros de entrenamiento

Parámetro	Descripción	Valor
img	tamaño de la imagen	640
batch	tamaño del lote	16
epochs	épocas de entrenamiento	100
data	la ruta del archivo yaml	Enfermedades-de-Cacao-1/data.yaml
weights	el archivo de ruta de pesos pre-entrenados	yolov5s.pt

Fuente: Elaboración propia

La configuración elegida para el entrenamiento se realizó a base de las diferentes pruebas y recomendaciones de los creadores de YOLOv5, se aprovechó el uso del transfer learning para optimizar el uso de recursos al crear nuestro modelo. Los resultados se pueden visualizar a continuación.

Ilustración 27 Resultados del entrenamiento



Fuente: Elaboración propia

En la tabla 4 se detallan los valores finales de las métricas más importantes.

Tabla 4 Resultados de las métricas

Métricas	Valor
mAP 0.5	0.8059
mAP 0.5:0.95	0.6619
Precision	0.8861
Recall	0.7337

Fuente: Elaboración propia

2.5. Ejecución y/o ensamblaje del prototipo

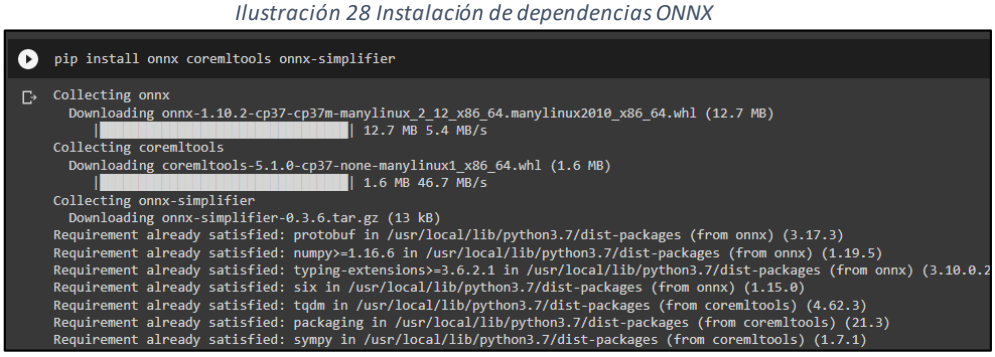
Esta sección abarca la ejecución y ensamblaje del prototipo final, para ello se realizaron dos actividades, primero la conversión del modelo y luego desarrollo de la aplicación móvil.

2.5.1. Conversión del modelo

Dado que se utilizará el framework NCNN de Tencent para utilizar nuestro modelo de detección en un entorno móvil, debemos seguir ciertos pasos para convertir nuestro modelo al formato requerido. Las herramientas para la conversión al formato ONNX fueron instaladas en Google Colaboratory mientras que las herramientas de NCNN en un entorno Linux.

1. Instalamos las dependencias para convertir nuestro modelo al formato ONNX (Intercambio de redes neuronales abiertas).

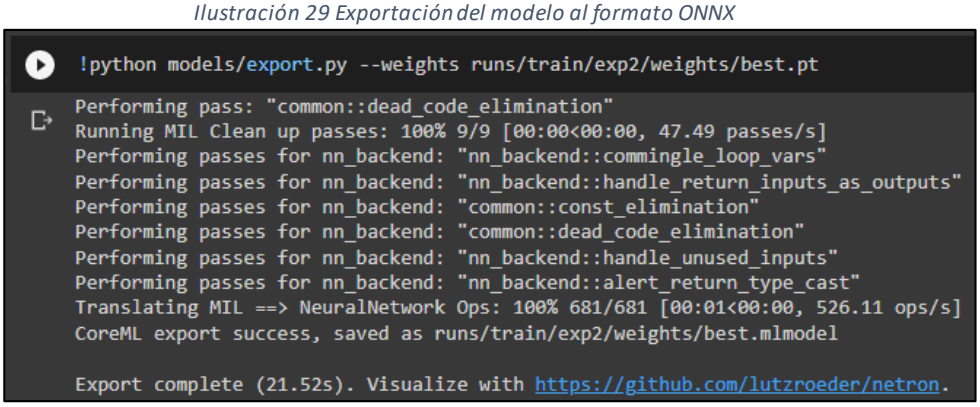
Ilustración 28 Instalación de dependencias ONNX



Fuente: Elaboración propia

2. Ejecutamos la exportación de nuestro modelo entrenado al formato ONNX.

Ilustración 29 Exportación del modelo al formato ONNX



Fuente: Elaboración propia

3. Realizamos la simplificación del modelo ONNX previamente obtenido.

Ilustración 30 Simplificación del modelo ONNX

```
!python -m onnxsim runs/train/exp2/weights/best.onnx runs/train/exp2/weights/best-sim.onnx

Simplifying...
Checking 0/3...
Checking 1/3...
Checking 2/3...
Ok!
```

Fuente: Elaboración propia

4. Debido a que las herramientas de NCNN necesitan la librería protobuf, realizamos su descarga desde su repositorio.

Ilustración 31 Descarga de protobuf

```
raul@ruilova: ~/Escritorio/Titulacion
raul@ruilova:~/Escritorio/Titulacion$ git clone https://github.com/protocolbuffers/protobuf.git
Clonando en 'protobuf'...
remote: Enumerating objects: 93268, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 93268 (delta 0), reused 3 (delta 0), pack-reused 93262
Recibiendo objetos: 100% (93268/93268), 85.28 MiB | 665.00 KiB/s, listo.
Resolviendo deltas: 100% (65204/65204), listo.
Actualizando archivos: 100% (2449/2449), listo.
```

Fuente: Elaboración propia

5. Configuramos protobuf siguiendo la documentación encontrada en el repositorio.

Ilustración 32 Configuración de protobuf

```
raul@ruilova: ~/Escritorio/Titulacion/protobuf
raul@ruilova:~/Escritorio/Titulacion$ cd protobuf
raul@ruilova:~/Escritorio/Titulacion/protobuf$ git submodule update --init --recursive
raul@ruilova:~/Escritorio/Titulacion/protobuf$ ./autogen.sh
+ test -d third_party/googletest
+ mkdir -p third_party/googletest/m4
+ autoreconf -f -i -Wall,no-obsolete
libtoolize: putting auxiliary files in AC_CONFIG_AUX_DIR, 'build-aux'.
libtoolize: copying file 'build-aux/ltmain.sh'
libtoolize: putting macros in AC_CONFIG_MACRO_DIRS, 'm4'.
libtoolize: copying file 'm4/libtool.m4'
```

Fuente: Elaboración propia

6. Realizamos la descarga del repositorio del framework NCNN que contiene todos los archivos necesarios para trabajar la conversión del modelo.

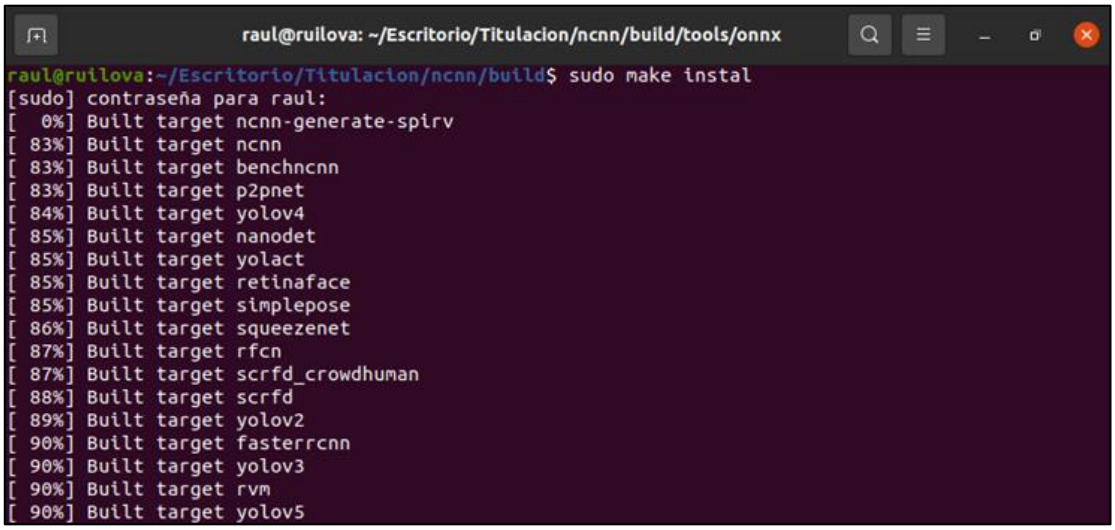
Ilustración 33 Descarga de NCNN

```
raul@ruilova: ~/Escritorio/Titulacion
raul@ruilova:~/Escritorio/Titulacion$ git clone https://github.com/Tencent/ncnn.git
Clonando en 'ncnn'...
remote: Enumerating objects: 24013, done.
remote: Counting objects: 100% (631/631), done.
remote: Compressing objects: 100% (244/244), done.
remote: Total 24013 (delta 439), reused 510 (delta 387), pack-reused 23382
Recibiendo objetos: 100% (24013/24013), 16.88 MiB | 708.00 KiB/s, listo.
Resolviendo deltas: 100% (19978/19978), listo.
Actualizando archivos: 100% (2382/2382), listo.
```

Fuente: Elaboración propia

7. Instalamos las herramientas y dependencias de NCNN.

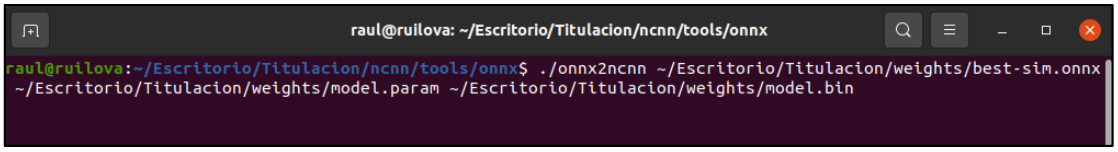
Ilustración 34 Instalación de NCNN



Fuente: Elaboración propia

6. Con las dependencias instaladas podemos ejecutar la conversión del modelo onnx simplificado con las herramientas de NCNN.

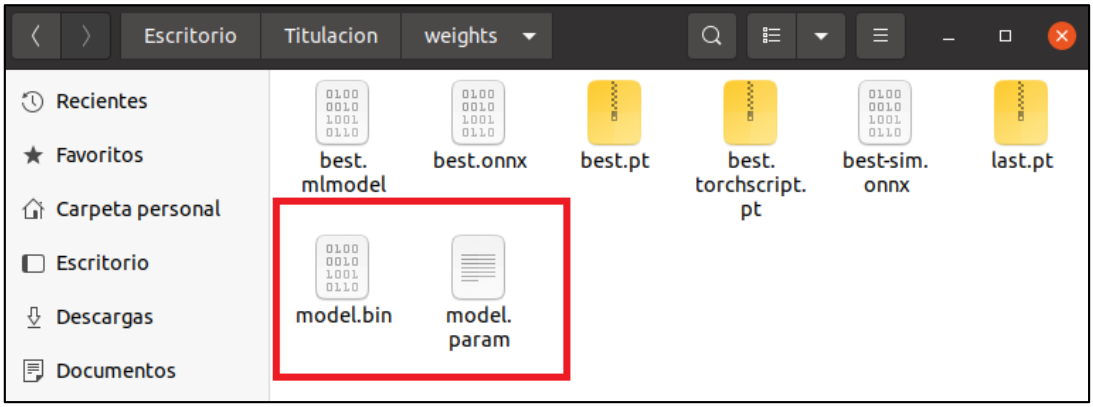
Ilustración 35 Ejecución de la conversión con las herramientas de NCNN



Fuente: Elaboración propia

7. Obtenemos los archivos resultantes de nuestro modelo que consta de un archivo de extensión bin y param.

Ilustración 36 Obtención de los archivos del modelo convertido



Fuente: Elaboración propia

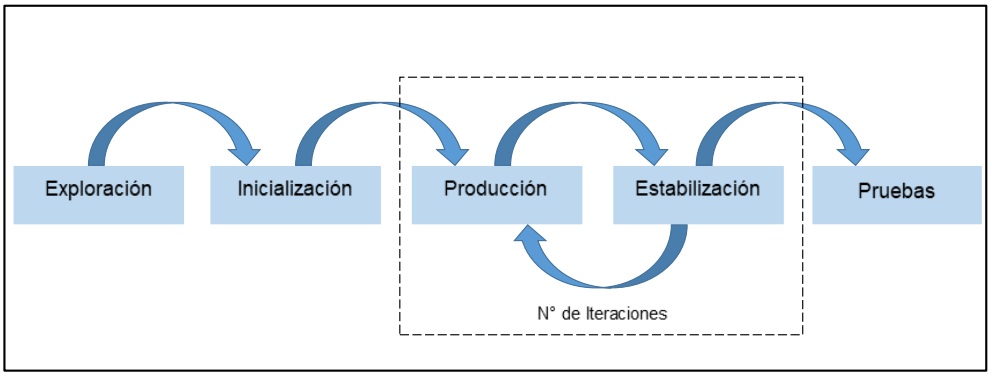
2.5.2. Desarrollo de la aplicación móvil

Para superar con éxito los retos que representa el desarrollo móvil, se siguió la metodología de desarrollo Mobile-D orientada al desarrollo de aplicaciones móviles.

2.5.2.1. Mobile-D

Mobile-D es una metodología ágil diseñada para trabajar en un formato optimizado, con un grupo pequeño de desarrollo y en un plazo máximo de diez semanas para la entrega del producto final. Cuenta con las siguientes fases: exploración, inicialización, producción, estabilización y pruebas. [48]

Ilustración 37 Fases de la metodología Mobile-D



Fuente: Elaboración propia

2.5.2.1.1. Exploración

En esta fase se definen los interesados, así como requisitos funcionales y no funcionales para el desarrollo de la aplicación. Además del establecimiento del alcance de la aplicación

2.5.2.1.1.1. Interesados del proyecto

La siguiente tabla detalla las partes interesadas en el proyecto de forma directa o indirecta.

Tabla 5 Interesados del proyecto

N°	Interesado	Descripción
1	Usuarios de la aplicación	Pequeños productores de cacao
2	Desarrolladores	Autor del presente proyecto

Fuente: Elaboración propia

2.5.2.1.1.2. Requisitos funcionales

Los requisitos funcionales describen las diferentes funciones que el sistema es capaz de realizar.

Tabla 6 RF01 – Seleccionar foto de la galería

Especificación de requerimiento funcional			
Código	RF01	**Prioridad**	Alta
Nombre	Seleccionar foto de la galería		
Fecha		**Responsable**	Raúl Ruilova
Descripción	El usuario podrá acceder a su galería y seleccionar una foto para su detección		
Actores	Usuario		
Pre-Condición	-		
Post-Condición	-		

Fuente: Elaboración propia

Tabla 7 RF02 - Tomar foto de la cámara

Especificación de requerimiento funcional			
Código	RF02	**Prioridad**	Alta
Nombre	Tomar foto de la cámara		
Fecha		**Responsable**	Raúl Ruilova
Descripción	El usuario podrá abrir la cámara y tomar una foto para su detección		
Actores	Usuario		
Pre-Condición	-		
Post-Condición	-		

Fuente: Elaboración propia

Tabla 8 RF03 - Seleccionar clases

Especificación de requerimiento funcional			
Código	RF03	**Prioridad**	Alta
Nombre	Seleccionar clases		
Fecha		**Responsable**	Raúl Ruilova
Descripción	El usuario podrá seleccionar las clases que desea detectar		
Actores	Usuario		
Pre-Condición	-		
Post-Condición	-		

Fuente: Elaboración propia

Tabla 9 RF04 - Activar y/o desactivar GPU

Especificación de requerimiento funcional			
Código	RF04	**Prioridad**	Alta
Nombre	Activar y/o desactivar GPU		
Fecha		**Responsable**	Raúl Ruilova
Descripción	El usuario podrá activar y/o desactivar el uso del GPU a la hora de realizar la detección		
Actores	Usuario		
Pre-Condición	-		
Post-Condición	-		

Fuente: Elaboración propia

Tabla 10 RF05 - Ejecutar detección

Especificación de requerimiento funcional			
Código	RF05	**Prioridad**	Alta
Nombre	Ejecutar detección		
Fecha		**Responsable**	Raúl Ruilova
Descripción	El usuario podrá realizar la detección de las clases (fitóftora, monilia y sana) en la foto seleccionada y se guardara automáticamente		
Actores	Usuario		
Pre-Condición	-		
Post-Condición	-		

Fuente: Elaboración propia

Tabla 11 RF06 - Ver Galería

Especificación de requerimiento funcional			
Código	RF06	**Prioridad**	Media
Nombre	Ver Galería		
Fecha		**Responsable**	Raúl Ruilova
Descripción	El usuario accederá a la Galería de la aplicación para poder ver todas las imágenes detectadas		
Actores	Usuario		
Pre-Condición	-		
Post-Condición	-		

Fuente: Elaboración propia

Tabla 12 RF07 - Compartir foto

Especificación de requerimiento funcional			
Código	RF07	**Prioridad**	Media
Nombre	Compartir foto		
Fecha		**Responsable**	Raúl Ruilova
Descripción	El usuario podrá seleccionar la foto que quiere compartir		
Actores	Usuario		
Pre-Condición	Debe existir una foto tomada y/o detectada		
Post-Condición	-		

Fuente: Elaboración propia

Tabla 13 RF08 - Renombrar foto

Especificación de requerimiento funcional			
Código	RF08	**Prioridad**	Media
Nombre	Renombrar foto		
Fecha		**Responsable**	Raúl Ruilova
Descripción	El usuario podrá seleccionar la foto que quiere renombrar		
Actores	Usuario		
Pre-Condición	Debe existir una foto tomada y/o detectada		
Post-Condición	-		

Fuente: Elaboración propia

Tabla 14 RF09 - Eliminar foto

Especificación de requerimiento funcional			
Código	RF09	**Prioridad**	Media
Nombre	Eliminar foto		
Fecha		**Responsable**	Raúl Ruilova
Descripción	El usuario podrá seleccionar la foto que quiere eliminar		
Actores	Usuario		
Pre-Condición	Debe existir una foto tomada y/o detectada		
Post-Condición	-		

Fuente: Elaboración propia

2.5.2.1.1.3. Requisitos no funcionales

Los requisitos no funcionales no están relacionados con el aspecto funcionalidad del sistema, pero si se relacionan a las cualidades del mismo.

Tabla 15 Requisitos no funcionales de la aplicación móvil

Código	Requisito	Descripción	Prioridad
RNF01	Plataforma	La aplicación podrá ser utilizada en dispositivos móviles Android	Alta
RNF02	Interfaz de usuario	La interfaz de usuario debe ser amigable con el usuario y fácil de usar	Alta
RNF03	Sistema de navegación	El usuario podrá navegar a través de las ventanas de la aplicación (Menú principal, Detección y Galería)	Alta

Fuente: Elaboración propia

2.5.2.1.1.4. Definición del Alcance

El alcance incluye la detección de dos de las principales enfermedades que afectan a la mazorca del cacao en Ecuador, la monilia y la fitóftora, las cuales generan la pudrición de estas, además de detectar las mazorcas que se encuentren en un estado saludable.

Por ende, se desarrolla una aplicación móvil que hará uso de la cámara del smartphone para capturar fotos, además de poder seleccionar de la propia galería. La foto será procesada y detectará las mazorcas de modo que serán encerradas en un cuadro delimitador indicando si el fruto está siendo afectado o no.

2.5.2.1.2. Inicialización

Esta fase se definen los recursos tecnológicos de hardware y software para el desarrollo de la aplicación. Además de la planificación de las fases y el diseño de las interfaces.

2.5.2.1.2.1. Hardware

Los recursos de hardware son los diferentes dispositivos que se usaron para el desarrollo de la aplicación móvil.

Tabla 16 Recursos de hardware

Computador	Windows 10 Pro
	Procesador Intel(R) Core(TM) i7-3770
	8 GB de Memoria RAM
Dispositivo móvil	Android 10
	Procesador Snapdragon 665 de 8 núcleos
	4 GB de Memoria RAM
	64 GB de almacenamiento

Fuente: Elaboración propia

2.5.2.1.2.2. Software

Los recursos de software son los programas utilizados para el desarrollo de la aplicación móvil.

Tabla 17 Recursos de software

Nombre	Descripción	Versión
Android Studio	Entorno de desarrollo integrado	4.0.1
LDPlayer4	Emulador	4.0.42

Fuente: Elaboración propia

2.5.2.1.2.3. Planificación

La planificación conlleva cada una de las etapas y las tareas a realizar divididas en iteraciones.

Tabla 18 Planificación de fases

Fase	Iteración	Descripción
Exploración	Iteración 0	Establecimiento de los interesados del proyecto y requisitos
Inicialización	Iteración 0	Descripción de los recursos tecnológicos y planificación
Producción	Iteración 1	Implementación de la interfaz del módulo de detección
	Iteración 2	Implementación de la funcionalidad del módulo de detección

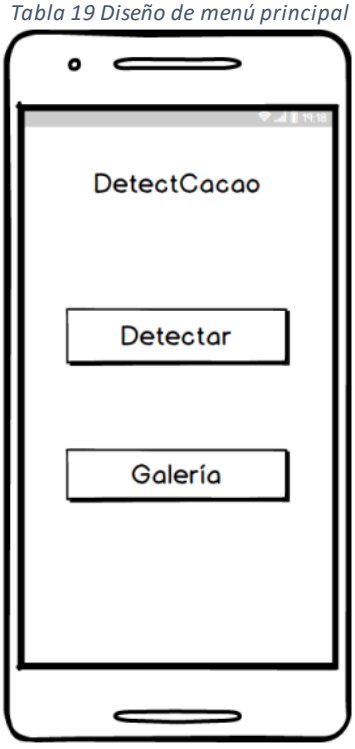
	Iteración 3	Implementación de la interfaz del módulo de galería
	Iteración 4	Implementación de la funcionalidad del módulo de galería
Estabilización	Iteración 5	Refactorización de la funcionalidad del módulo de detección
	Iteración 6	Refactorización de la funcionalidad del módulo de galería
Pruebas	Iteración 7	Aplicación de pruebas a la aplicación y análisis de resultados.

Fuente: Elaboración propia

2.5.2.1.2.4. Diseño de interfaces

Menú Principal

Tabla 19 Diseño de menú principal

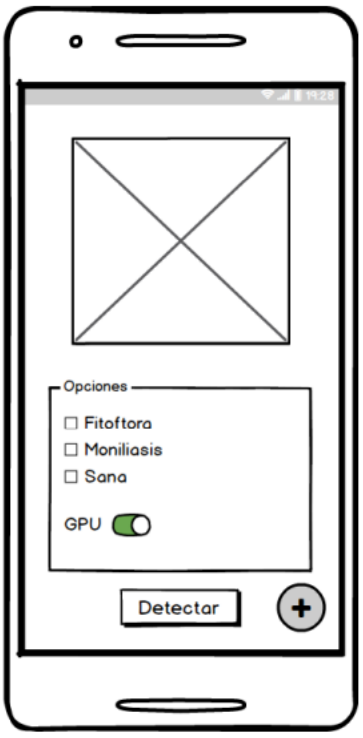


Fuente: Elaboración propia

Para el menú principal se consideró una interfaz sencilla conformada por el nombre de la aplicación y el menú de opciones que dan acceso al módulo de detección y al de galería.

Módulo de detección

Tabla 20 Diseño de la interfaz de detección

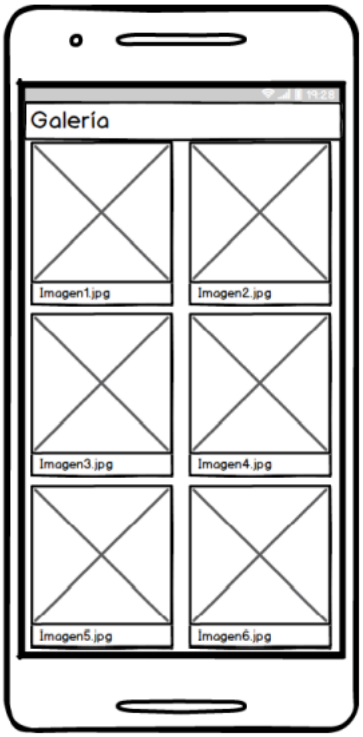


Fuente: Elaboración propia

El módulo de detección se conforma de una única interfaz donde se podrá detectar las imágenes. Se conforma del contenedor de la imagen y un menú de opciones para seleccionar las clases y activar y/o desactivar el GPU. Además de un botón flotando para generar una foto ya sea por medio de la cámara y/o galería y el botón para ejecutar la detección.

Módulo de Galería

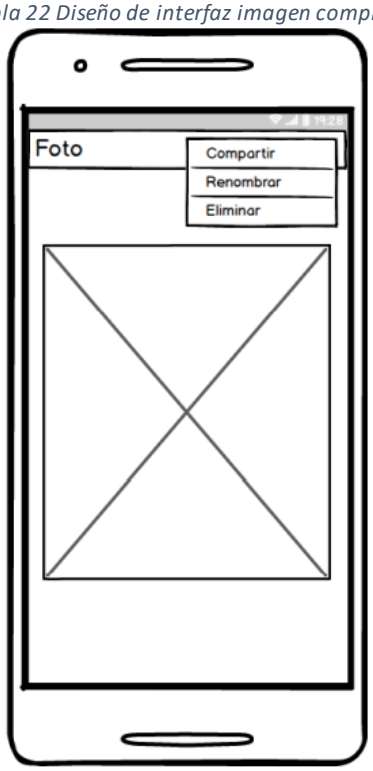
Tabla 21 Diseño de la interfaz de galería



Fuente: Elaboración propia

El módulo de Galería está conformado en primera instancia de una cuadrícula de imágenes, las cuales contienen la pre visualización del a misma y el nombre.

Tabla 22 Diseño de interfaz imagen completa



Fuente: Elaboración propia

Al seleccionar una imagen se abrirá una interfaz que contendrá la imagen completa y un menú desplegable con las opciones de: compartir, renombrar y eliminar.

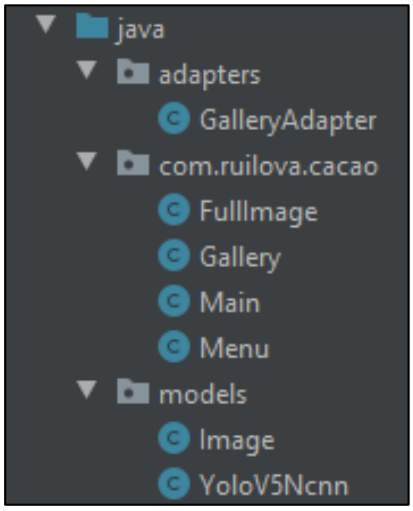
2.5.2.1.3. Producción y Estabilización

Tanto la fase de producción como la de estabilización buscan realizar las funcionalidades de la aplicación móvil basándose en los requisitos recolectados en las fases anteriores.

2.5.2.1.3.1. Estructura del proyecto

La estructura de directorios del proyecto contiene tres secciones donde se encuentran las clases que conforman el proyecto. El primer directorio denominado con el identificador del proyecto "com.ruilova.cacao" posee las clases para las ventanas del menú principal, módulo de detección, galería y ver imagen. El directorio del "adapters" posee el adaptador utilizado para la Galería. Finalmente tenemos los modelos utilizados en el directorio "models", estos corresponden a las clases Image para la galería y Yolov5Ncnn para las detecciones.

Ilustración 38 Estructura del proyecto



Fuente: Elaboración propia

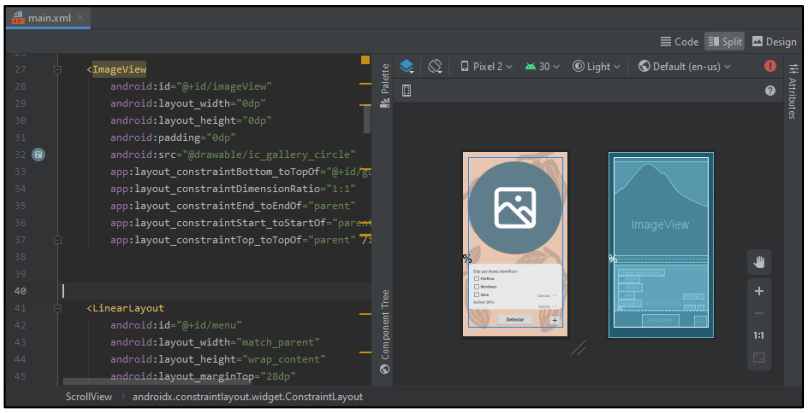
A continuación, se detallan los módulos que componen la aplicación.

2.5.2.1.3.2. Módulo de detección

Esta ventana corresponde a la sección donde el usuario realizará la selección de imágenes para la detección de las enfermedades. Para ello primero se diseñó

una interfaz que posee elementos como un imageview para visualizar la imagen seleccionada y/o detectada, un cardview para contener los cuadros de selección de las clases y switch del gpu, un botón para realizar la detección y un botón flotante con opciones para seleccionar la imagen desde la galería y/o cámara.

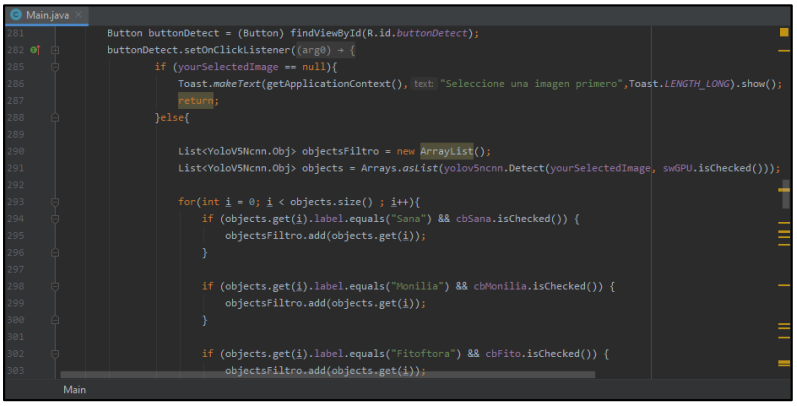
Ilustración 39 Interfaz gráfica del módulo de detección



Fuente: Elaboración propia

En la clase se realizó la programación de las diferentes funcionalidades, primero se desarrolló el botón de detectar el cual toma la foto presente en el imageview para procesarla mediante la clase Yolov5Cnn, la cual arroja mediante el framework una lista de objetos detectados.

Ilustración 40 Función de detección



Fuente: Elaboración propia

La función “showObjects” dibuja un cuadrado en cada objeto previamente obtenido y junto con una clase y su valor de confianza.

Ilustración 41 Función para mostrar los objetos

Fuente: Elaboración propia

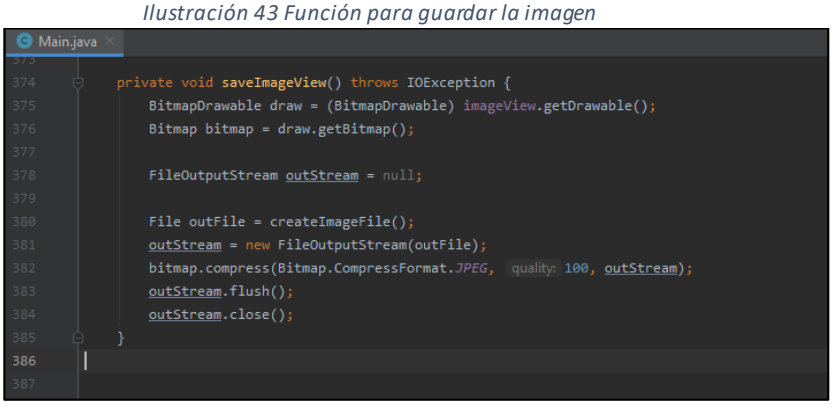
La clase Yolov5Cnn como se muestra en la ilustración 38, posee los atributos para las coordenadas del cuadro delimitador, la etiqueta de la clase y la confianza. Además de la función que carga la librería del Framework.

Ilustración 42 Clase Yolov5Cnn

Fuente: Elaboración propia

Con “saveImageView” se realiza el guardado de la foto detectada mediante la creación de un archivo de imagen en formato jpg.

Ilustración 43 Función para guardar la imagen



Fuente: Elaboración propia

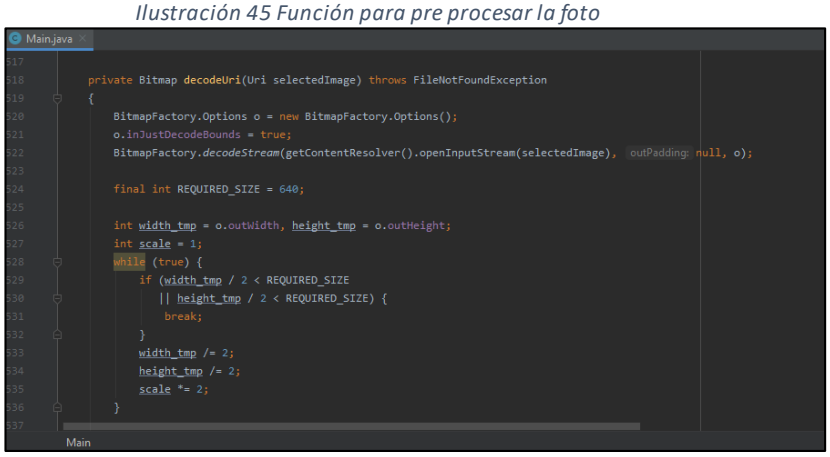
Para realizar la toma de fotos desarrollamos primero una función para pedir los permisos de la cámara y segundo una función para realizar la captura de la foto y la creación del archivo.

Ilustración 44 Función para tomar foto desde la cámara

Fuente: Elaboración propia

Para asegurar que las fotos seleccionadas se visualicen correctamente en la interfaz se realizó una función para aplicar un pre procesado, realizando una redimensión y ajuste de la orientación de la imagen.

Ilustración 45 Función para pre procesar la foto

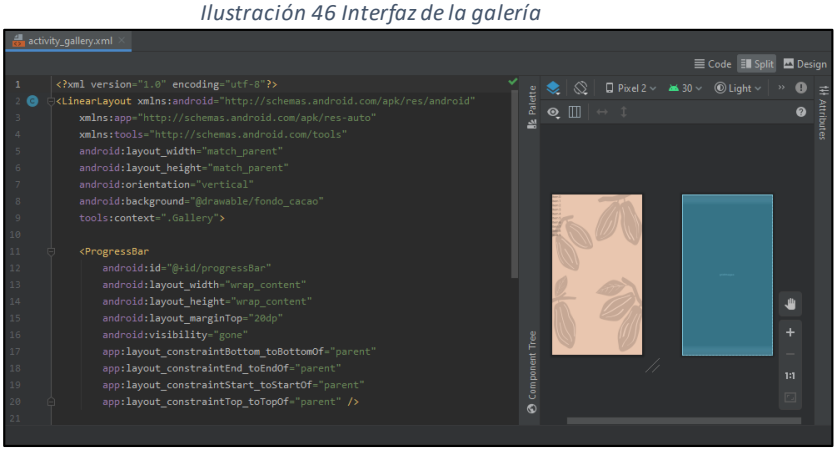


Fuente: Elaboración propia

2.5.2.1.3.3. Módulo galería

La galería muestra cada una de las fotos que se han tomado y realizado la detección. Además de poder seleccionar por independiente las imágenes y obtener una vista completa de la imagen. Primero se desarrolló la interfaz que muestra la totalidad de las fotos, esta se compone de un recyclerview

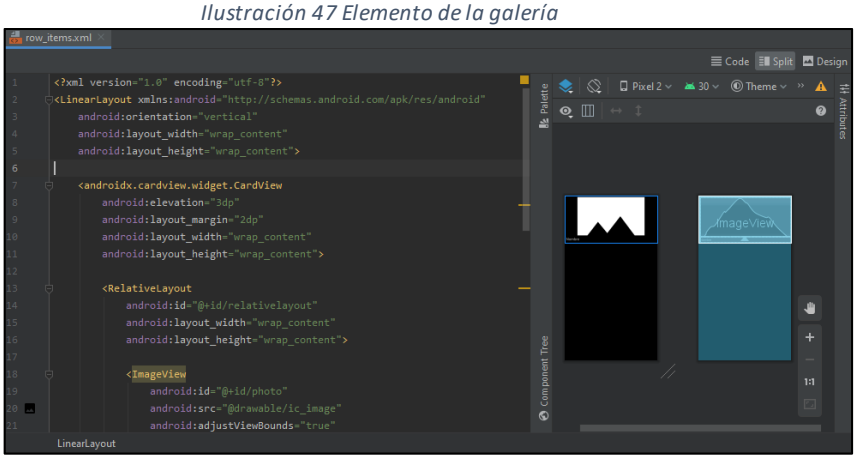
Ilustración 46 Interfaz de la galería



Fuente: Elaboración propia

Además, se creó el elemento cardview que contendrá la imagen y el nombre de cada foto.

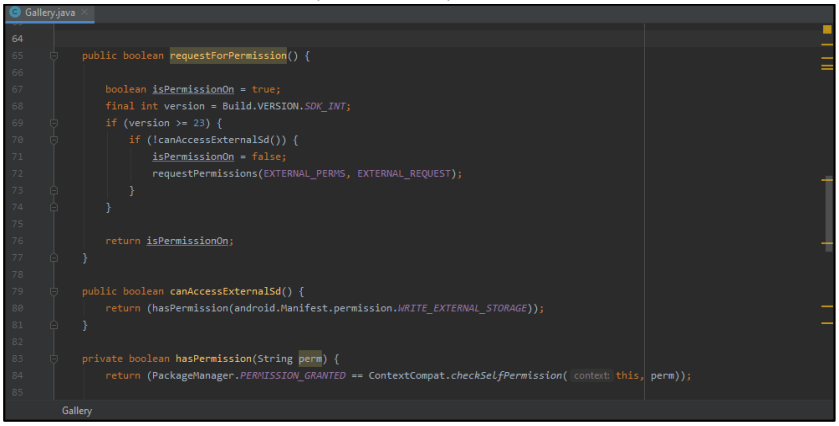
Ilustración 47 Elemento de la galería



Fuente: Elaboración propia

En la clase principal de la Galería se desarrolló la función para solicitar los permisos para acceder al almacenamiento del teléfono.

Ilustración 48 Función para obtener permisos de acceder al almacenamiento



Fuente: Elaboración propia

Así mismo se realiza una función que obtiene los archivos de la carpeta que la aplicación utiliza para almacenar las fotos. Por cada foto creamos un objeto de tipo image y lo añadimos a una colección.

Ilustración 49 Función para obtener las fotos de la aplicación

```
Gallery.java
88
89      public void getFromFolder()
90      {
91          File[] listFile;
92          String storageDir = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES)+"/CacaoApp";
93
94          File file = new File(storageDir);
95
96          if (file.isDirectory())
97          {
98              listFile = file.listFiles();
99
100             if (listFile != null) {
101                 for (int i = 0; i < listFile.length; i++)
102                 {
103                     images.add(new Image(listFile[i].getName(),listFile[i].getAbsolutePath()));
104
105                 }
106             }
107         }
108     }
109
Gallery
```

Fuente: Elaboración propia

La clase image posee los atributos correspondientes al nombre y dirección de la foto, además de los métodos para mostrar.

Ilustración 50 Clase Image

```
Image.java
1   package models;
2
3   public class Image {
4       private String name;
5       private String photo;
6
7       public Image(String name, String photo) {
8           this.name = name;
9           this.photo = photo;
10      }
11
12      public String getName() { return name; }
15
16      public String getPhoto() { return photo; }
19  }
```

Fuente: Elaboración propia

En el “OnCreate” definimos el adaptador, en este adaptador pasamos la colección de imágenes. Con el administrador de diseño establecemos el número de cuadrículas a dos por fila.

Ilustración 51 Función OnCreate de la Galería

Fuente: Elaboración propia

En la clase del Adaptador tenemos tres métodos que lo constituyen, el primero es para crear un viewholder y devolverlo. Aquí establecemos que el diseño de nuestra vista será el layout del cardview.

Ilustración 52 Función onCreateViewHolder del Adaptador

Fuente: Elaboración propia

En el segundo método enlazamos el viewholder y creamos los procedimientos para mostrar la imagen y el texto. Además del evento clic en el layout.

Ilustración 53 Función nonBindViewHolder() del Adaptador

Fuente: Elaboración propia

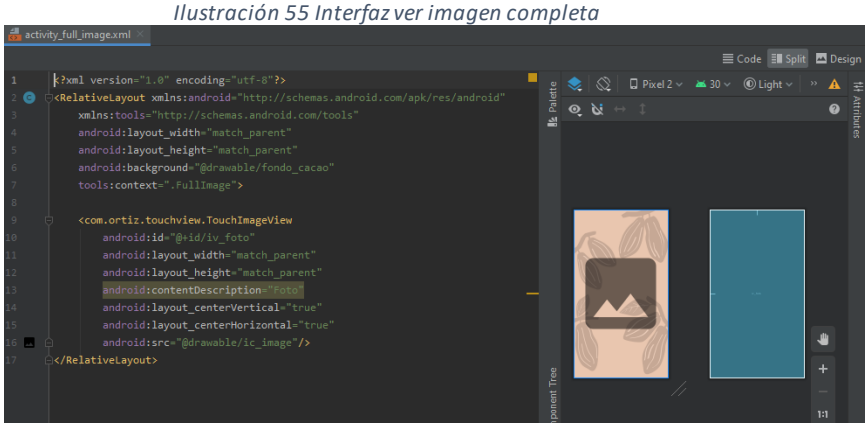
Por último, la función “getItemCount” que contiene el tamaño de la lista, además de la función que da soporte al recyclerview para definir todos los elementos de la interfaz.

Ilustración 54 Función getItemCount() del Adaptador

Fuente: Elaboración propia

Para la ventana de ver la imagen completa, se diseñó una interfaz con un elemento touchview para poder obtener la función de ampliar la imagen.

Ilustración 55 Interfaz ver imagen completa



Fuente: Elaboración propia

En clase se programó una función para ajustar la orientación de la imagen y establecerla correctamente en el imageview.

Ilustración 56 Función para preprocesar la imagen completa

Fuente: Elaboración propia

Además de la creación del menú de opciones y las respectivas funcionalidades. Como lo son: compartir, renombrar y eliminar imagen.

Ilustración 57 Menú de opciones y funciones

Fuente: Elaboración propia

2.5.2.1.4. Pruebas

Para las pruebas se constató el cumplimiento de las vistas necesarias para la construcción de la aplicación móvil.

Tabla 23 Verificación de vistas

Módulo	Vista	Cumplimento
General	Interfaz menú	Hecho
Detección	Interfaz de detección	Hecho
Galería	Interfaz de galería	Hecho
	Interfaz imágen completa	Hecho

Fuente: Elaboración propia

Además de las funcionalidades que se dictaminaron durante el establecimiento de los requisitos.

Tabla 24 Verificación de funcionalidades

Módulo	Funcionalidad	Cumplimento
Detección	Tomar foto de la cámara	Hecho
	Seleccionar clases	Hecho
	Activar y/o desactivar GPU	Hecho
	Ejecutar detección	Hecho
Galería	Ver Galería	Hecho
	Compartir foto	Hecho
	Renombrar foto	Hecho
	Eliminar foto	Hecho

Fuente: Elaboración propia

3. CAPÍTULO III: EVALUACIÓN DEL PROTOTIPO

3.1. Plan de Evaluación

Para realizar la evaluación del prototipo, se realiza un análisis de rendimiento del modelo YOLOv5 resultante y el cual es ejecutado en la aplicación móvil. Para esta tarea se tomarán las imágenes de la carpeta test del dataset que se generó en las etapas tempranas del desarrollo del prototipo y proceder a ejecutar la detección en cada imagen.

Los resultados obtenidos en las detecciones se registrarán, y finalmente se tomará como herramienta evaluadora la matriz de confusión, utilizada en el campo de la inteligencia artificial para calcular las métricas de modelos.

2.5.3. Matriz de confusión

La matriz de confusión, es una matriz simple de 2x2 que compara dos estados opuestos, donde las columnas representan la realidad ontológica (teoría verdadera o no verdadera), mientras que las filas representan la realidad empírica (basado en datos empíricos que confirman la presencia o no del caso de estudio). [49] En nuestro trabajo se considera la realidad ontológica como la situación real del estado de la mazorca de cacao (fitóftora, monilia, sana), mientras que la realidad empírica es la predicción obtenida por nuestra aplicación.

Tabla 25 Estructura de la matriz de confusión

		Predicción	
		Objeto Presente	Objeto no Presente
Situación Real	Objeto Presente	Verdadero Positivo (VP)	Falso Negativo (FN)
	Objeto no Presente	Falso Positivo (FP)	Verdadero Negativo (VN)

Fuente: Elaboración propia

Los cuatros valores presentes en las celdas centrales son los posibles estados del cruce con la realidad ontológica y la realidad empírica y se detallan a continuación.

- **Verdadero positivo. -** Representa una situación en la que la teoría es verdadera y la evidencia lo confirma
- **Falso positivo. -** Representa una situación en la que la teoría es falsa pero los datos empíricos obtenidos nos llevan a creer que la teoría es verdadera
- **Falso negativo. -** Representa una situación que la teoría es verdadera pero los datos empíricos no lo confirman
- **Verdadero negativo. -** Representa una situación en que la teoría es falsa y la evidencia lo confirma.

2.5.4. Métricas

En la detección de objetos se pueden asociar métricas propias como la precisión media promedio (mAP) y para la clasificación como la precisión (precision), la sensibilidad (recall), y el valor de referencia (f-score).

Precisión

Es la medida de datos predichos correctamente de los valores positivos. La fórmula se define por la relación de los verdaderos positivos y el total de datos predichos como positivos.

$$Precisión = \frac{Verdadero\ Positivo}{Verdadero\ Positivo + Falso\ Positivo}$$

Sensibilidad

Es la medida que hace referencia a la capacidad del modelo de obtener valores positivos. Su fórmula es la relación de verdaderos positivos y el total de positivos reales.

$$Sensibilidad = \frac{Verdadero\ Positivo}{Verdadero\ Positivo + Falso\ Negativo}$$

Valor de Referencia

Se define como la media armónica de la precisión y la sensibilidad del modelo. El valor más alto es la medida de 1.0 lo que significa una precisión y sensibilidad perfectas por parte del modelo. [50]

$$Valor\ de\ Referencia = \frac{2 * Precisión * Sensibilidad}{Precisión + Sensibilidad}$$

Precisión media promedio

Es la métrica principal de precisión en la detección de objetos que valora la puntuación del área detectada del objeto. Con todas las predicciones se establece una curva de recuperación de precisión y el área bajo la curva se define como la precisión media (AP). Para obtener el promedio se calcula el AP de todas las clases. [51]

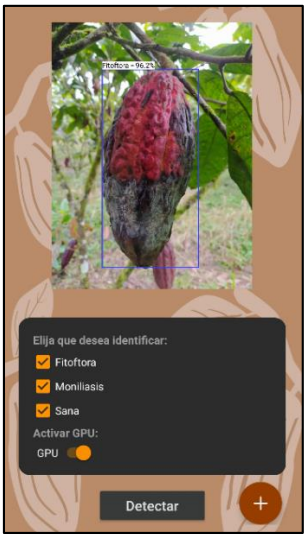
2.5.5. Resultados de la Evaluación

La evaluación se realizó con una cantidad de 129 imágenes que son las predefinidas en el dataset. Cada imagen fue detectada en la aplicación móvil y se generaron las matrices de confusión para cada clase, como resultado se obtuvieron los cálculos de las métricas de clasificación.

Matriz de confusión de la clase Fitóftora

En la siguiente imagen se evidencia la detección de una imagen con una mazorca enferma de fitóftora.

Ilustración 58 Detección de fitóftora en la aplicación



Fuente: Elaboración propia

Para la clase fitóftora se evaluaron un total de 76 mazorcas de cacao. Donde 55 fueron detectadas correctamente y 26 incorrectamente por la aplicación.

Tabla 26 Matriz de confusión de la clase fitóftora

Fitóftora		Predicción	
		Objeto Presente	Objeto no Presente
Situación Real	Objeto Presente	55	26
	Objeto no Presente	14	332

Fuente: Elaboración propia

Matriz de confusión de la clase Monilia

En la siguiente imagen se evidencia la detección de una imagen con una mazorca enferma de monilia.

Ilustración 59 Detección de monilia en la aplicación

Fuente: Elaboración propia

En la clase monilia se evaluaron un total de 49 mazorcas de cacao. Como se puede visualizar en la siguiente matriz 45 fueron detectadas correctamente y tan solo 4 incorrectamente.

Tabla 27 Matriz de confusión de la clase monilia

Monilia		Predicción	
		Objeto Presente	Objeto no Presente
Situación Real	Objeto Presente	45	4
	Objeto no Presente	11	367

Fuente: Elaboración propia

Matriz de confusión de la clase Sana

A continuación, se evidencia la detección de una imagen con dos mazorcas sanas.

Ilustración 60 Detección de sana en la aplicación

Fuente: Elaboración propia

En la clase sana se evaluaron un total de 284 mazorcas de cacao. La evaluación obtuvo un total de 259 detectadas correctamente.

Tabla 28 Matriz de confusión de la clase sana

Sana		**Predicción**	
		Objeto Presente	Objeto no Presente
Situación Real	Objeto Presente	259	25
	Objeto no Presente	14	129

Fuente: Elaboración propia

Resultados del modelo

Con las matrices de confusión realizadas para cada clase, se calcularon las métricas de clasificación individualmente para cada clase y luego se calculó un promedio total. En la siguiente tabla se detallan los resultados obtenidos de las métricas de precisión, sensibilidad y valor de referencia para las clases fitóftora, monilia y sana.

Tabla 29 Resultados de evaluación

Clase	Precisión	Sensibilidad	Valor de Referencia
Fitóftora	0.80	0.68	0.73
Monilia	0.80	0.92	0.86
Sana	0.95	0.91	0.93
Total	0.85	0.84	0.84

Fuente: Elaboración propia

Los resultados indican que nuestro modelo tiene valores aceptables para la clasificación de mazorcas enfermas y resultados excelentes para las no enfermas; los valores obtenidos para la clase fitóftora fueron de precisión de 0.80, sensibilidad de 0.68 y valor de referencia de 0.73, para la clase monilia; precisión de 0.80, sensibilidad de 0.92 y valor de referencia de 0.86, para la clase sana; precisión de 0.95, sensibilidad 0.91 y valor de referencia de 0.93. Esto significa que nuestra aplicación móvil es capaz de detectar todas las clases a un nivel suficiente siempre y cuando se evalúen imágenes de buena calidad y donde se aprecien las mazorcas completas.

3.2. Conclusiones

Como resultado del desarrollo de la aplicación móvil para la detección de enfermedades en la mazorca de cacao utilizando deep learning se puede concluir que:

- La literatura menciona el uso del deep learning en diferentes campos donde se pueden abordar problemas del mundo real, como es el caso de la detección de objetos. En la actualidad existen varios algoritmos y técnicas que permiten generar modelos de alta calidad a través de las redes neuronales convolucionales siendo una de las tecnologías que se encuentra a la vanguardia.
- El procedimiento llevado a cabo a la hora de formar el conjunto de datos, fue el considerar herramientas como LabelImg y Roboflow, además de técnicas como el aumento de datos y el etiquetado de imágenes con el fin de obtener una óptima construcción del dataset.
- A través de los resultados obtenidos por el entrenamiento del modelo con YOLOV5 en su versión pequeña se obtiene un mAP del 0.81, precisión del 0.89 y sensibilidad del 0.73. Al ejecutar las pruebas se evaluó únicamente con métricas de clasificación obteniendo así resultados de precisión del 0.85, sensibilidad del 0.84 y valor de referencia del 0.84. Mediante un análisis se concluye que el modelo cumple con las expectativas, teniendo un nivel aceptable tanto para la localización como para la clasificación de mazorcas enfermas y saludables.
- El diseño del prototipo se consideró un desafío que se pudo superar con la aplicación de una metodología ágil donde se desarrolló de forma eficiente la aplicación móvil con todas sus requisitos e interfaces. Además del uso del framework Tencent NCNN que hizo posible la ejecución del modelo de deep learning en un ámbito móvil.

3.3. Recomendaciones

- Se recomienda mejorar la cantidad de imágenes para entrenar un modelo de deep learning mejor y más completo, de manera que la aplicación móvil pueda detectar correctamente todas las mazorcas enfermas.
- Añadir imágenes de mazorcas del cacao nacional y de esta manera garantizar el reconocimiento correcto de este tipo de cacao.
- Realizar el entrenamiento con una mejor GPU, y así escalar los valores de los parámetros de entrenamiento con el fin de obtener mejores resultados en las métricas finales.
- Implementar funcionalidades de visión por computadora para realizar la detección de mazorcas con video en tiempo real.
- Desarrollar la aplicación móvil para el sistema operativo iOS y de esta forma abarcar una mayor cantidad de usuarios.

4. BIBLIOGRAFÍA

[1] T. Liu, B. Pang, L. Zhang, W. Yang, y X. Sun, «Sea Surface Object Detection Algorithm Based on YOLO v4 Fused with Reverse Depthwise Separable Convolution (RDSC) for USV», *J. Mar. Sci. Eng.*, vol. 9, n.º 7, p. 753, jul. 2021, doi: 10.3390/jmse9070753.

[2] M. L. Mekhalfi, C. Nicolo, Y. Bazi, M. M. A. Rahhal, N. A. Alsharif, y E. A. Maghayreh, «Contrasting YOLOv5, Transformer, and EfficientDet Detectors for Crop Circle Detection in Desert», *IEEE Geosci. Remote Sens. Lett.*, vol. 19, pp. 1-5, 2022, doi: 10.1109/LGRS.2021.3085139.

[3] ANECACAO, «Sector Exportador de Cacao», 2019. http://www.anecacao.com/index.php/es/estadisticas/estadisticas-actuales.html (accedido 23 de noviembre de 2021).

[4] El Telégrafo, «National Geographic destacó a Ecuador como un gran productor de cacao», 15 de noviembre de 2020. https://www.eltelegrafo.com.ec/noticias/actualidad/44/national-geographic-ecuador-gran-productor-cacao (accedido 21 de noviembre de 2021).

[5] Z. K. Solís Hidalgo, S. L. Peñaherrera Villafuerte, y D. I. Vera Coello, *Las enfermedades del cacao y las buenas prácticas agronómicas para su manejo*. Mocache, EC: INIAP, Estación Experimental Tropical Pichilingue, 2021. Accedido: 21 de noviembre de 2021. [En línea]. Disponible en: http://repositorio.iniap.gob.ec/handle/41000/5747

[6] C. Meske, E. Bunde, J. Schneider, y M. Gersch, «Explainable Artificial Intelligence: Objectives, Stakeholders, and Future Research Opportunities», *Inf. Syst. Manag.*, pp. 1-11, dic. 2020, doi: 10.1080/10580530.2020.1849465.

[7] X. Jing, P. Peng, y Z. Huang, «Analysis of multi-level capital market linkage driven by artificial intelligence and deep learning methods», *Soft Comput.*, vol. 24, n.º 11, pp. 8011-8019, jun. 2020, doi: 10.1007/s00500-019-04095-z.

[8] S. S. Alahmari, D. B. Goldgof, P. R. Mouton, y L. O. Hall, «Challenges for the Repeatability of Deep Learning Models», *IEEE Access*, vol. 8, pp. 211860-211868, 2020, doi: 10.1109/ACCESS.2020.3039833.

[9] R. Y. Montesino, J. A. Rosales-Huamani, y J. L. Castillo-Sequera, «Detection of phytophthora palmivora in cocoa fruit with deep learning», en *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*, Chaves, Portugal, jun. 2021, pp. 1-4. doi: 10.23919/CISTI52073.2021.9476279.

[10] Basri, R. Tamin, H. A. Karim, Indrabayu, y I. S. Areni, «Mobile Image Processing Application for Cacao's Fruits Pest and Disease Attack Using Deep Learning Algorithm». ICIC International 学会, 2020. Accedido: 6 de enero de 2022. [En línea]. Disponible en: https://doi.org/10.24507/icicel.14.10.1025

[11] Y. Fang, X. Guo, K. Chen, Z. Zhou, y Q. Ye, «Accurate and automated detection of surface knots on sawn timbers using YOLO-V5 model», *BioResources*, vol. 16, n.º 3, pp. 5390-5406, jun. 2021, doi: 10.15376/biores.16.3.5390-5406.

[12] Ultralytics, «Tips for Best Training Results - YOLOv5 Documentation». https://docs.ultralytics.com/tutorials/training-tips-best-results/ (accedido 21 de enero de 2022).

[13] V. Anzules-Toala, E. Pazmiño-Bonilla, L. Alvarado-Huamán, R. Borjas-Ventura, V. Castro-Cepero, y A. Julca-Otiniano, «Control of cacao (Theobroma cacao) diseases in Santo Domingo de los Tsachilas, Ecuador», *Agron. Mesoam.*, pp. 45939-45939, 2022, doi: 10.15517/am.v33i1.45939.

[14] L. Zhu, X. Geng, Z. Li, y C. Liu, «Improving YOLOv5 with Attention Mechanism for Detecting Boulders from Planetary Images», *Remote Sens.*, vol. 13, n.º 18, p. 3776, sep. 2021, doi: 10.3390/rs13183776.

[15] S. C. Sennott, L. Akagi, M. Lee, y A. Rhodes, «AAC and Artificial Intelligence (AI)», *Top. Lang. Disord.*, vol. 39, n.º 4, pp. 389-403, oct. 2019, doi: 10.1097/TLD.0000000000000197.

[16] M. Xue, C. Yuan, H. Wu, Y. Zhang, y W. Liu, «Machine Learning Security: Threats, Countermeasures, and Evaluations», *IEEE Access*, vol. 8, pp. 74720-74742, 2020, doi: 10.1109/ACCESS.2020.2987435.

[17] D.-Y. Kim, J.-H. Park, Y. Lee, y S. Kim, «Network virtualization for real-time processing of object detection using deep learning», *Multimed. Tools Appl.*, vol. 80, n.º 28-29, pp. 35851-35869, nov. 2021, doi: 10.1007/s11042-020-09603-0.

[18] D. Cao, Z. Chen, y L. Gao, «An improved object detection algorithm based on multi-scaled and deformable convolutional neural networks», *Hum.-Centric Comput. Inf. Sci.*, vol. 10, n.º 1, p. 14, dic. 2020, doi: 10.1186/s13673-020-00219-9.

[19] S. Bera y V. K. Shrivastava, «Effect of pooling strategy on convolutional neural network for classification of hyperspectral remote sensing images», *IET Image Process.*, vol. 14, n.º 3, pp. 480-486, feb. 2020, doi: 10.1049/iet-ipr.2019.0561.

[20] J. Gu *et al.*, «Recent advances in convolutional neural networks», *Pattern Recognit.*, vol. 77, pp. 354-377, may 2018, doi: 10.1016/j.patcog.2017.10.013.

[21] S. Ray, K. Alshouiliy, y D. P. Agrawal, «Dimensionality Reduction for Human Activity Recognition Using Google Colab», *Information*, vol. 12, n.º 1, p. 6, dic. 2020, doi: 10.3390/info12010006.

[22] B. Prashanth, M. Mendu, y R. Thallapalli, «Cloud based Machine learning with advanced predictive Analytics using Google Colaboratory», *Mater. Today Proc.*, p. S221478532100897X, mar. 2021, doi: 10.1016/j.matpr.2021.01.800.

[23] F. Jubayer *et al.*, «Detection of mold on the food surface using YOLOv5», *Curr. Res. Food Sci.*, vol. 4, pp. 724-728, 2021, doi: 10.1016/j.crfs.2021.10.003.

[24] «OpenCV: Introducción a los tutoriales de OpenCV-Python». https://docs.opencv.org/4.x/d0/de3/tutorial_py_intro.html (accedido 27 de diciembre de 2021).

[25] H. Choi y J. Lee, «Efficient Use of GPU Memory for Large-Scale Deep Learning Model Training», *Appl. Sci.*, vol. 11, n.º 21, p. 10377, nov. 2021, doi: 10.3390/app112110377.

[26] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, y M. Shafique, «Hardware and Software Optimizations for Accelerating Deep Neural Networks: Survey of Current Trends, Challenges, and the Road Ahead», *IEEE Access*, vol. 8, pp. 225134-225180, 2020, doi: 10.1109/ACCESS.2020.3039858.

[27] Y. Wang *et al.*, «Benchmarking the Performance and Energy Efficiency of AI Accelerators for AI Training», en *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, Melbourne, Australia, may 2020, pp. 744-751. doi: 10.1109/CCGrid49817.2020.00-15.

[28] F. Zhang, Z. Chen, C. Zhang, A. C. Zhou, J. Zhai, y X. Du, «An Efficient Parallel Secure Machine Learning Framework on GPUs», *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, n.º 9, pp. 2262-2276, sep. 2021, doi: 10.1109/TPDS.2021.3059108.

[29] «Very short-term spatial and temporal wind power forecasting: A deep learning approach», *CSEE J. Power Energy Syst.*, ago. 2019, doi: 10.17775/CSEEJPES.2018.00010.

[30] S. Madichetty y S. Muthukumarasamy, «Detection of situational information from Twitter during disaster using deep learning models», *Sādhanā*, vol. 45, n.º 1, p. 270, dic. 2020, doi: 10.1007/s12046-020-01504-0.

[31] J. Wang, T. Zhang, Y. Cheng, y N. Al-Nabhan, «Deep Learning for Object Detection: A Survey», *Comput. Syst. Sci. Eng.*, vol. 38, n.º 2, pp. 165-182, 2021, doi: 10.32604/csse.2021.017016.

[32] J. Yao, J. Qi, J. Zhang, H. Shao, J. Yang, y X. Li, «A Real-Time Detection Algorithm for Kiwifruit Defects Based on YOLOv5», *Electronics*, vol. 10, n.º 14, p. 1711, jul. 2021, doi: 10.3390/electronics10141711.

[33] S. Srivastava, A. V. Divekar, C. Anilkumar, I. Naik, V. Kulkarni, y V. Pattabiraman, «Comparative analysis of deep learning image detection algorithms», *J. Big Data*, vol. 8, n.º 1, p. 66, dic. 2021, doi: 10.1186/s40537-021-00434-w.

[34] Y.-M. Chang, W.-C. Liao, S.-C. Wang, C.-C. Yang, y Y.-S. Hwang, «A framework for scheduling dependent programs on GPU architectures», *J. Syst. Archit.*, vol. 106, p. 101712, jun. 2020, doi: 10.1016/j.sysarc.2020.101712.

[35] D. Danopoulos, C. Kachris, y D. Soudris, «Utilizing cloud FPGAs towards the open neural network standard», *Sustain. Comput. Inform. Syst.*, vol. 30, p. 100520, jun. 2021, doi: 10.1016/j.suscom.2021.100520.

[36] Y. Y. Liau y K. Ryu, «Status Recognition Using Pre-Trained YOLOv5 for Sustainable Human-Robot Collaboration (HRC) System in Mold Assembly», *Sustainability*, vol. 13, n.º 21, p. 12044, oct. 2021, doi: 10.3390/su132112044.

[37] N. D. Nath, A. H. Behzadan, y S. G. Paal, «Deep learning for site safety: Real-time detection of personal protective equipment», *Autom. Constr.*, vol. 112, p. 103085, abr. 2020, doi: 10.1016/j.autcon.2020.103085.

[38] M. Rivero-Palacio, W. Alfonso-Morales, y E. Caicedo-Bravo, «Mobile Application for Anemia Detection through Ocular Conjunctiva Images», en *2021 IEEE Colombian Conference on Applications of Computational Intelligence (ColCACI)*, Cali, Colombia, may 2021, pp. 1-6. doi: 10.1109/ColCACI52978.2021.9469593.

[39] F. Lin, T. Hou, Q. Jin, y A. You, «Improved YOLO Based Detection Algorithm for Floating Debris in Waterway», *Entropy*, vol. 23, n.º 9, p. 1111, ago. 2021, doi: 10.3390/e23091111.

[40] C. Shorten y T. M. Khoshgoftaar, «A survey on Image Data Augmentation for Deep Learning», *J. Big Data*, vol. 6, n.º 1, p. 60, dic. 2019, doi: 10.1186/s40537-019-0197-0.
[41] darrenl, *LabelImg*. 2021. Accedido: 28 de diciembre de 2021. [En línea]. Disponible en: https://github.com/tzutalin/labelImg
[42] «Roboflow». https://roboflow.ai (accedido 28 de diciembre de 2021).
[43] «Introducción a Android Studio», *Android Developers*. https://developer.android.com/studio/intro?hl=es-419 (accedido 29 de diciembre de 2021).
[44] U. Farooq, Z. Zhao, M. Sridharan, y I. Neamtiu, «LiveDroid: identifying and preserving mobile app state in volatile runtime environments», *Proc. ACM Program. Lang.*, vol. 4, n.º OOPSLA, pp. 1-30, nov. 2020, doi: 10.1145/3428228.
[45] «¿Qué es Java y por qué lo necesito?» https://www.java.com/en/download/help/whatis_java.html (accedido 4 de enero de 2022).
[46] *ncnn*. Tencent, 2022. Accedido: 2 de enero de 2022. [En línea]. Disponible en: https://github.com/Tencent/ncnn
[47] «Cocoa Diseases (YOLOv4)». https://kaggle.com/serranosebas/enfermedades-cacao-yolov4 (accedido 29 de diciembre de 2021).
[48] M. Gamboa-Ramos, R. Gómez-Noa, O. Iparraguirre-Villanueva, M. Cabanillas-Carbonell, y J. L. H. Salazar, «Mobile Application with Augmented Reality to Improve Learning in Science and Technology», *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, n.º 10, 2021, doi: 10.14569/IJACSA.2021.0121055.
[49] B. Befani, «Quality of quality: A diagnostic approach to qualitative evaluation», *Evaluation*, vol. 26, n.º 3, pp. 333-349, jul. 2020, doi: 10.1177/1356389019898223.
[50] Md. M. Hossain *et al.*, «Prediction on Domestic Violence in Bangladesh during the COVID-19 Outbreak Using Machine Learning Methods», *Appl. Syst. Innov.*, vol. 4, n.º 4, p. 77, oct. 2021, doi: 10.3390/asi4040077.
[51] D. Dlužnevskij, P. Stefanovič, y S. Ramanauskaitė, «Investigation of YOLOv5 Efficiency in iPhone Supported Systems», *Balt. J. Mod. Comput.*, vol. 9, n.º 3, 2021, doi: 10.22364/bjmc.2021.9.3.07.