UNIVERSIDAD NACIONAL AUTÓNOMA DE NICARAGUA

UNAN-LEÓN

ÁREA DE CONOCIMIENTO DE CIENCIAS Y TECNOLOGÍA.

ÁREA ESPECÍFICA DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN.

MONOGRAFÍA PARA OPTAR AL TÍTULO DE INGENIERÍA EN SISTEMAS DE LA INFORMACIÓN

IMPLEMENTACIÓN DE UNA INTELIGENCIA ARTIFICIAL A TRAVÉS DE UNA RED NEURONAL CONVOLUCIONAL PARA LA DETECCIÓN DE PLAGAS Y ENFERMEDADES EN EL CULTIVO DE ARROZ

Autores (as)

- Br. Jackson José Montenegro Gómez
- Br. Jefferson Antoine Wilson Torres
- Br. Sergio Abdiel Duarte Castillo

Tutor (a)

- M. Sc. Wilmer Matamoros

León, marzo de 2024

¡45/19 La Patria, La Revolución!

UNIVERSIDAD NACIONAL AUTÓNOMA DE NICARAGUA

UNAN-LEÓN

ÁREA DE CONOCIMIENTO DE CIENCIAS Y TECNOLOGÍA.

ÁREA ESPECÍFICA DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN.

MONOGRAFÍA PARA OPTAR AL TÍTULO DE INGENIERÍA EN SISTEMAS DE LA INFORMACIÓN

IMPLEMENTACIÓN DE UNA INTELIGENCIA ARTIFICIAL A TRAVÉS DE UNA RED NEURONAL CONVOLUCIONAL PARA LA DETECCIÓN DE PLAGAS Y ENFERMEDADES EN EL CULTIVO DE ARROZ

Autores (as)

- Br. Jackson José Montenegro Gómez
- Br. Jefferson Antoine Wilson Torres
- Br. Sergio Abdiel Duarte Castillo

Tutor (a)

- M. Sc. Wilmer Matamoros

León, marzo de 2024

¡45/19 La Patria, La Revolución!

ÍNDICE

1. CAPÍTULO I: INTRODUCCIÓN

La implementación de una Inteligencia Artificial (IA) a través de una red neuronal convolucional (RNC) para la detección de plagas y enfermedades en el cultivo de arroz es una técnica prometedora y de gran importancia en el ámbito agrícola.

La creación de una RNC para la detección de plagas y enfermedades en el cultivo de arroz implica entrenar el modelo con una amplia variedad de imágenes que representen diferentes tipos de plagas y enfermedades presentes en este cultivo. Una vez entrenado, el modelo puede ser utilizado para analizar imágenes en tiempo real y detectar la presencia de plagas y enfermedades en las plantas de arroz.

Esta tecnología de IA brinda múltiples ventajas, como la detección temprana de plagas y enfermedades, lo cual permite una respuesta rápida y eficiente para su control. Además, reduce la dependencia de inspecciones manuales y la necesidad de utilizar productos químicos de manera indiscriminada, lo que puede tener un impacto negativo en el medio ambiente.

El principal alcance está enfocado a los agricultores y empresas productoras de arroz en el departamento de León, Nicaragua. Tendrán una aplicación que le ayudará a mejorar la producción de arroz, detectando las enfermedades o plagas que afecten a su cultivo de arroz de manera rápida y eficaz, a través de un diagnóstico, dando una oportunidad al productor de actuar rápido y evitar que se propaguen.

1.1. Antecedentes

“Aplicación de Inteligencia Artificial en la Predicción y Monitoreo de Enfermedades en Cultivos”, elaborado por María González Pérez, Universidad Agraria de Buenos Aires, Argentina, julio de 2021. Esta monografía examina el uso de técnicas de inteligencia artificial, incluyendo algoritmos de aprendizaje automático, para predecir y monitorear enfermedades en cultivos agrícolas. Se explora la aplicación de modelos predictivos basados en datos históricos y en tiempo real para mejorar la respuesta a epidemias agrícolas, con un enfoque especial en la optimización del uso de pesticidas y recursos hídricos.

“Automatización y Robótica Agrícola: Un Enfoque Impulsado por la Inteligencia Artificial”, elaborado por Juan Martínez López, Universidad Politécnica de Valencia, España, septiembre de 2020. Esta investigación aborda la integración de sistemas de inteligencia artificial en la robótica agrícola para lograr una mayor eficiencia en tareas como la siembra, cosecha y mantenimiento de cultivos. Se examina el uso de algoritmos de visión por computadora y aprendizaje profundo para la identificación de malas hierbas, optimizando así la aplicación de herbicidas y mejorando la productividad agrícola.

Las investigaciones previas destacan el impacto positivo de la inteligencia artificial en la agricultura, con énfasis en la optimización de procesos y la toma de decisiones agrícolas. En el contexto actual, nos alineamos con estas tendencias, demostrando cómo la IA contribuye a la detección eficiente de enfermedades en cultivos como el arroz, mejorando la gestión agrícola.

1.2. Planteamiento del problema

El sector agrícola se enfrenta constantemente a desafíos relacionados con la salud de los cultivos, como la proliferación de plagas y enfermedades que afectan negativamente la producción y calidad de los alimentos. La detección temprana de estas plagas y enfermedades es crucial para tomar medidas rápidas y efectivas que minimicen los daños y pérdidas. Es por eso por lo que nos hacemos las siguientes preguntas.

PREGUNTA GENERAL

¿Es posible desarrollar un sistema para la detección de plagas y enfermedades en el cultivo de arroz utilizando redes neuronales convolucionales?

PREGUNTAS ESPECÍFICAS

¿Cómo establecer las funcionalidades principales del sistema?

¿Por qué medio se realizará la detección de plagas y enfermedades en el cultivo de arroz?

¿Dónde se implementará la información detectada?

¿Es necesaria la creación de una aplicación móvil? ¿Por qué?

1.3. Justificación

Hoy en día Nicaragua es un país donde la agricultura forma la principal actividad económica debido a que los principales cultivos, como el arroz, que se establecen son generadores de ingresos y permiten la subsistencia de las diferentes familias del campo, de igual manera contribuye a la seguridad alimentaria. (Jiménez, 2017).

Estamos claros que la agricultura es muy importante en Nicaragua, la problemática se presenta a la hora de cosechar. Donde pueden influir muchos aspectos, como la selección de la semilla, la preparación de la tierra, el monitoreo, entre otras. Son una de las prácticas fundamentales para poder tener cultivos saludables. Los nicaragüenses realizan la práctica de la agricultura de manera tradicional, donde estos conocimientos pasan de generación en generación, pero muchas veces estas prácticas no son muy efectivas, las cuales podemos identificar para mejorarlas y aumentar la producción. La inteligencia artificial nos puede ayudar a simplificar este trabajo, asimismo, garantizar una mejor efectividad y eficiencia en la productividad de las cosechas.

Nuestro proyecto “Implementación de una Inteligencia Artificial a través de una red neuronal convolucional para la detección de plagas y enfermedades en el cultivo de arroz”, puede permitirnos demostrar las potencialidades de la Inteligencia Artificial, que cada vez nos sigue impresionando más las capacidades que puede llegar a alcanzar. Presentándonos la resolución de problemas en las tareas de diferentes áreas, en este caso, poder detectar enfermedades o plagas que afectan cultivo de arroz. Esta forma de captar información nos puede ayudar a estar preparados para combatirlas, de este modo, evitamos que se propaguen o que afecte demasiado a la cosecha. Es la capacidad de estar preparados para la resolución de este tipo de problemas.

1.4. Objetivos

1.4.1. Objetivo General

- Desarrollar un sistema para la detección de plagas y enfermedades en el cultivo de arroz utilizando redes neuronales convolucionales.

1.4.2. Objetivos Específicos

- Definir los requerimientos por medio de métodos de recolección de datos para establecer las funcionalidades principales del sistema.

- Perfeccionar un modelo de Inteligencia Artificial basado en redes neuronales convolucionales, especializado en la identificación de plagas y enfermedades en cultivos de arroz.

- Implementar una aplicación backend dedicada a la gestión integral de la información necesaria para el sistema, asegurando un suministro eficiente y oportuno de datos requeridos por la aplicación.

- Desarrollar una aplicación móvil con la capacidad de capturar imágenes de cultivos de arroz afectados por plagas y enfermedades.

2. CAPÍTULO II: MARCO TEÓRICO.

2.1. Inteligencia Artificial

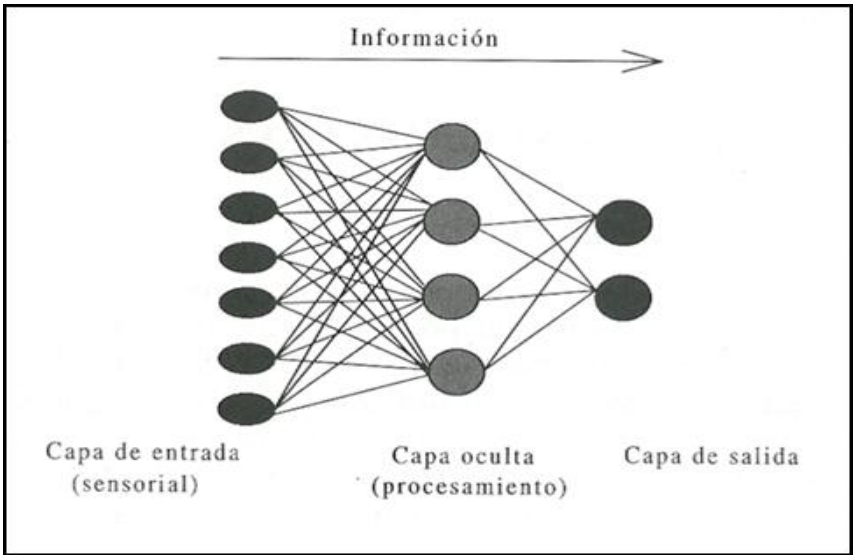
La inteligencia artificial (IA) es un campo de la informática que se enfoca en la creación de sistemas y algoritmos que pueden realizar tareas que, en general, requieren inteligencia humana, como el reconocimiento de voz, la toma de decisiones, el aprendizaje automático y la visión por computadora. La IA implica el uso de técnicas como el procesamiento del lenguaje natural, el aprendizaje automático y la minería de datos para crear sistemas que puedan simular la inteligencia humana y realizar tareas de manera autónoma.

2.1.1. Redes Neuronales

Las redes neuronales son una técnica de aprendizaje automático inspirada en la forma en que el cerebro humano procesa y analiza la información. Están compuestas por una serie de nodos interconectados que se asemejan a las neuronas del cerebro humano. Estos nodos procesan y transmiten la información a través de las conexiones entre ellos.

Cada nodo en una red neuronal procesa la información que recibe de sus entradas y luego transmite su salida a otros nodos de la red. La salida de un nodo se calcula mediante una función de activación que toma la suma ponderada de sus entradas y la transforma en una salida. Los pesos que se utilizan para calcular la suma ponderada se ajustan durante el entrenamiento de la red neuronal para mejorar su capacidad para realizar una tarea específica.

Las redes neuronales pueden tener varias capas de nodos interconectados, y se dividen en dos tipos principales: redes neuronales alimentadas hacia adelante y redes neuronales recurrentes. Las redes neuronales alimentadas hacia adelante son el tipo más común y se utilizan en aplicaciones como el reconocimiento de imágenes y el procesamiento del lenguaje natural. Las redes neuronales recurrentes son más complejas y se utilizan en aplicaciones que involucran datos secuenciales, como el procesamiento del habla y la traducción automática.



En esta imagen, los nodos en la capa de entrada representan las características de los datos de entrada, como píxeles en una imagen o palabras en una oración. Los nodos en la capa oculta realizan una transformación no lineal de las entradas y los nodos en la capa de salida producen la salida final de la red neuronal, como una clasificación o una predicción.

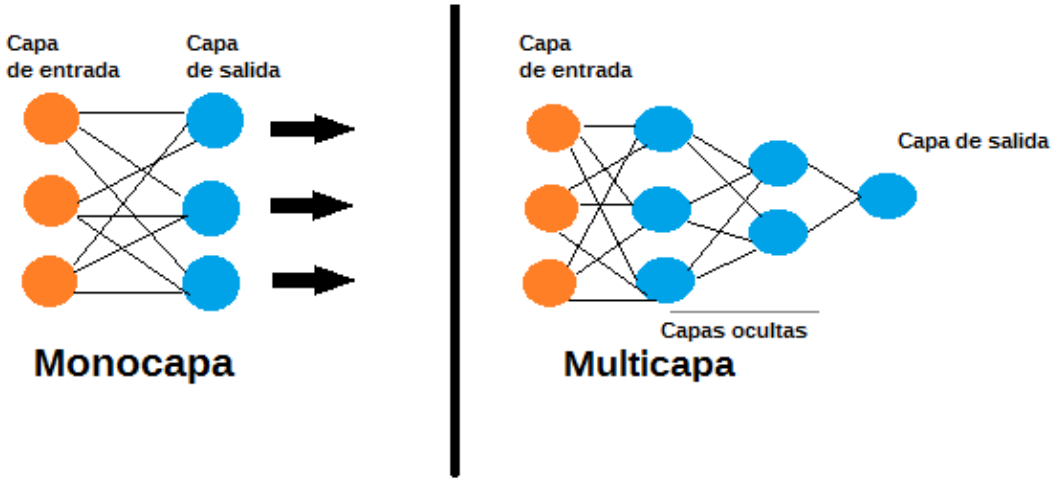
Durante el entrenamiento de la red neuronal, los pesos que se utilizan para calcular las salidas se ajustan mediante un algoritmo de optimización como el descenso del gradiente para minimizar la función de pérdida de la red. La función de pérdida mide cuánto se desvía la salida de la red neuronal de la salida deseada y se utiliza para guiar la optimización de los pesos.

2.1.2. Tipos de Redes Neuronales

En esta clasificación podemos dividir las redes neuronales, en dos principales tipos:

- **Redes neuronales monocapa**: Generalmente, son las más sencillas de hacer, la capa de entrada también podría considerarse una capa, pero al no hacer cálculos no se tiene en cuenta a la hora de clasificarse. La capa de entrada se conecta a la capa de neuronas de salida que realizan determinados cálculos.

- **Redes neuronales multicapas**: Entre las conexiones de entrada y de salida, existen diversas capas de neuronas que hacen de intermediarias, denominadas capas ocultas. Estas capas de neuronas pueden conectarse entre ellas o no.



2.1.2.1. Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales se distinguen de otras redes neuronales por su rendimiento superior con entradas de imagen, voz o señales de audio. Se componen de tres tipos principales de capas:

- Capa convolucional
- Capa de agrupación
- Capa totalmente conectada

Capa convolucional: Es el bloque de creación principal de una CNN, y es donde se realizan la mayoría de los cálculos. Requiere algunos componentes, como datos de entrada, un filtro y un mapa de características. Supongamos que la entrada es una imagen en color compuesta por una matriz de píxeles en 3D. Esto significa que la entrada tendrá tres dimensiones: altura, anchura y profundidad, que corresponden a la composición RGB en una imagen. También hay un detector de características, conocido como kernel o filtro, que se mueve por los campos receptivos de la imagen para comprobar si la característica está presente. Este proceso se denomina convolución.

Capa de agrupación: También conocida como submuestreo, permite reducir la dimensión mediante la reducción del número de parámetros de la entrada. De manera similar a la capa convolucional, la operación de agrupación barre toda la entrada con un filtro, pero la diferencia es que este filtro no tiene ningún peso. En su lugar, el kernel aplica una función de agregación a los valores dentro del campo receptivo y llena así la matriz de salida. Hay dos tipos principales de agrupación:

- **Agrupación máxima:** A medida que el filtro recorre la entrada, selecciona el píxel con el valor más alto para enviarlo a la matriz de salida. Este enfoque suele utilizarse más que la agrupación media.
- **Agrupación media:** Conforme el filtro avanza por la entrada, calcula el valor promedio dentro del campo receptivo para enviarlo a la matriz de salida. Aunque se pierde mucha información en la capa de agrupación, esta tiene una serie de beneficios para la CNN. Ayuda a reducir la complejidad, mejora la eficiencia y limita el riesgo de sobreajuste.

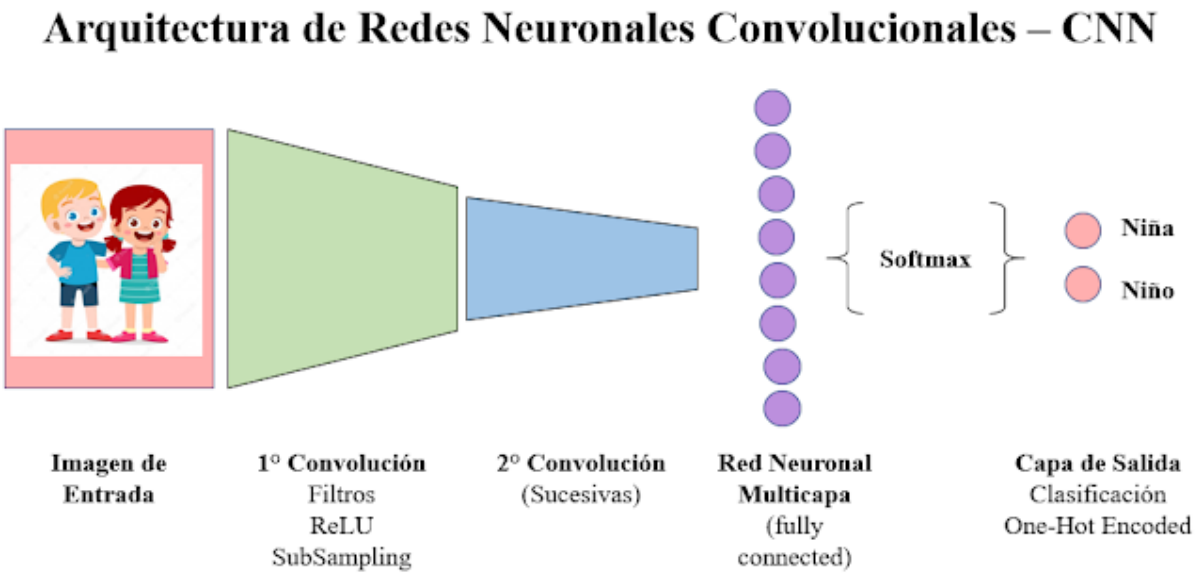
Capa totalmente conectada: Describe con precisión la capa en sí. Como se ha mencionado anteriormente, los valores de píxel de la imagen de entrada no están conectados directamente con la capa de salida en las capas parcialmente conectadas. Sin embargo, en la capa totalmente conectada, cada nodo de la capa de salida sí está conectado directamente a un nodo de la capa anterior. Esta capa realiza la tarea de clasificación basándose en las características extraídas de las capas anteriores y sus diferentes filtros.

La mayor ventaja de las redes neuronales convolucionales, respecto a otros tipos de redes neuronales como las multicapas comunes, es que diferentes partes de la red neuronal se pueden entrenar para tareas diversas. Con esto se consigue aumentar la velocidad de entrenamiento e identificar patrones de una forma más avanzada. Gracias al uso de algoritmos para redes convolucionales, se reduce el número de conexiones y de parámetros, por lo que requiere de menor entrenamiento.

2.1.2.2. ¿Cómo funcionan las Redes Neuronales Convolucionales?

Las redes convolucionales funcionan de manera opuesta al procesamiento de imágenes clásico basado en un algoritmo definido por humanos. Basándose en los datos de entrenamiento, la red neuronal convolucional extrae automáticamente características que luego se utilizarán para la clasificación de objetos.

Los campos de recepción local son un espacio separado en el que se buscan patrones. Las respuestas de los filtros de submuestreo y filtran repetidamente. Las neuronas de la corteza visual de los mamíferos están organizadas para procesar imágenes en capas. Algunos tienen una función para reconocer características locales como bordes, líneas o formas que se cruzan. Las claves utilizadas en el análisis tienen características específicas. La red neuronal convolucional se puede utilizar para determinar si la llave es cuadrada o redonda, qué tipo de agarre, agujero o qué forma es el perfil o color del diente.



2.2. Flutter

2.2.1. Definición

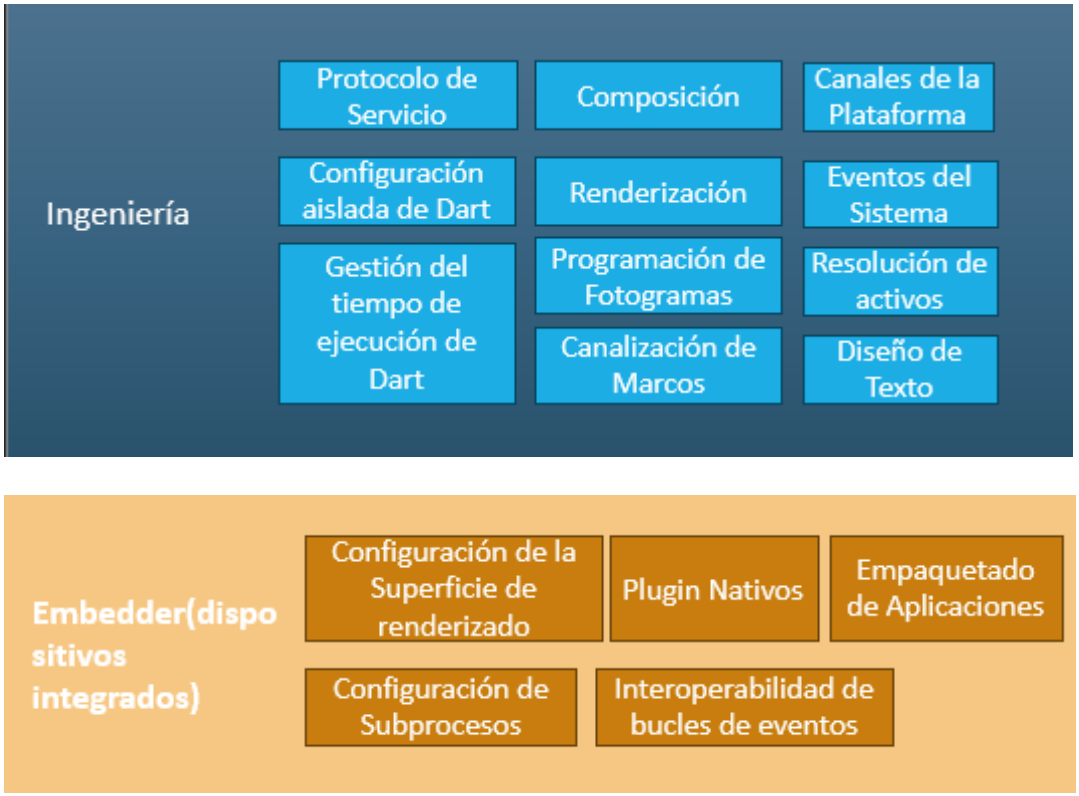
Flutter, se podría definir como un conjunto de herramientas de interfaz de usuario enfocada en el principio (DRY, Don't repeat yourself) o también conocido en la comunidad hispanohablante como (No te repetías!) , el cual nos da la idea de no hacer dos veces el mismo código, de este modo Flutter lleva este concepto a un nivel superior ya que el código escrito en Dart puede ser reutilizado a través de sistemas operativos como Android, IOS, Windows, Web (Wasm en prueba), Sistemas embebidos (Pantallas de automóviles, termostatos, frigoríficos modernos, y demás), MacOs, al mismo tiempo Flutter permite a las aplicaciones para interactuar directamente con los servicios que provee la plataforma en la cual está siendo utilizada.

El objetivo de Flutter es permitir a los desarrolladores ofrecer aplicaciones de alto rendimiento aprovechando las diferencias que provee cada sistema y plataforma mientras él mantiene un código que pueda ser reutilizado tanto como sea posible.

2.2.2. Arquitectura de Flutter

Flutter está construido como un sistema de capas extensibles. Existe como una serie de bibliotecas independientes, cada una de las cuales depende de la capa subyacente. Ninguna capa tiene acceso privilegiado a las capas inferiores y cada parte de la jerarquía del marco está diseñada para ser opcional y reemplazable.

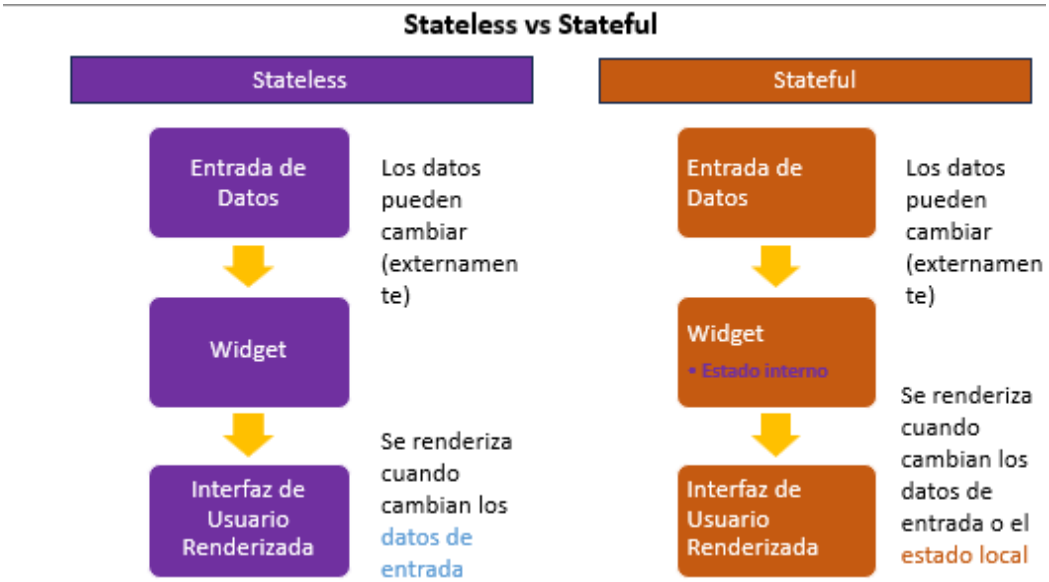
Las capas están clasificadas y definidas en las siguientes imágenes:



De las capa superior o Framework se pueden observar dos librerías principales, las cuales son Material y Cupertino, ambas capas están enfocadas en la apariencia enfocada a dispositivos móviles, aunque puedan ser utilizadas en otras plataformas,

Ambas son librerías que ya se encuentran en el núcleo del Framework, esto quiere decir que únicamente nos hace falta llamar la librería, tan fácil como hacer un import y empezar a utilizarlos.

En Flutter todo es un Widget, de los cuales son parte las dos clases base de un Widget, los StatelessWidget y los StatefulWidget:



También es importante por la parte de Flutter mencionar los manejadores de estados, y esto para que nos sirve en Flutter.

Se podría decir que un estado es la situación actual de nuestra app y en nuestro caso nuestros widgets, por ejemplo un usuario hace click en un checkbox y esto se representa como la situación o estado actual de nuestro widget, Otro ejemplo más común y útil, cuando hacemos una petición a nuestro backend mientras se realiza la petición podríamos activar un “LoadingStatus” que muestre en la parte de la UI un CircularProgressIndicator.

En el desarrollo de una aplicación normalmente existen dos tipos de estados.

1) Estado de la aplicación, es un estado no pasajero, es necesario mantenerlo a lo largo de la ejecución de nuestra aplicación ejemplos claves como preferencias de usuario (lenguaje, modo oscuro, estado de autentificación).
2) Estados efímeros a veces llamado como estado de la UI o estado de la vista actual, este estado puede ser comprendido de un solo widget, por ejemplo, un checkbox seleccionado, un botón (deshabilitado), página actual de un Bottom Navigation Bar.

Una vez aclarado esto es importante mencionar que en el ecosistema de Flutter hay varias opciones para manejar nuestros estados, mencionando los más

populares al momento de ser desarrollada nuestra aplicación móvil: BloC, Provider, GetX, Riverpod, Redux, InheritedWidget & InheritedModel. El gestor que hemos decidido usar es BloC.

2.2.3. ¿Por qué Flutter?

Utilizamos Flutter por que nos brinda diferentes ventajas a la hora de programar una aplicación, así también ventajas en la interacción con el usuario, algunas de estas que cabe destacar serían:

- Desarrollo multiplataforma eficiente: Flutter permite desarrollar aplicaciones para múltiples plataformas, como iOS, Android, web y escritorio, utilizando un único código base. Esto agiliza el proceso de desarrollo y mantenimiento, reduciendo la necesidad de escribir y mantener código separado para cada plataforma.
- Interfaz de usuario atractiva y personalizable: Flutter ofrece un amplio conjunto de widgets personalizables que permiten crear interfaces de usuario atractivas y consistentes. Su lenguaje de diseño, basado en widgets, facilita la creación de interfaces visuales ricas y dinámicas.
- Alto rendimiento y velocidad: Gracias a su arquitectura, Flutter ofrece un rendimiento de alta velocidad al utilizar el lenguaje de programación Dart y su motor gráfico, lo que permite que las aplicaciones sean fluidas y responsivas, incluso en dispositivos con recursos limitados.
- Hot Reload para desarrollo ágil: La función Hot Reload de Flutter permite a los desarrolladores ver los cambios realizados en tiempo real, lo que agiliza el proceso de desarrollo y facilita la corrección de errores de manera rápida y eficiente, mejorando la productividad del equipo de desarrollo.

2.3. Base de datos

En el mundo actual, la gestión eficiente de la información es esencial para el éxito de cualquier organización. En este contexto, las bases de datos juegan un papel fundamental al proporcionar un entorno estructurado y seguro para almacenar, organizar y recuperar datos de manera eficiente. Una opción destacada en este ámbito es PostgreSQL, un sistema de gestión de bases de datos relacional de código abierto que ha ganado reconocimiento por su confiabilidad, rendimiento y amplias capacidades.

2.3.1. PostgreSQL

PostgreSQL, a menudo referido como "PostgreSQL", ofrece un sólido conjunto de funciones que abarca desde la gestión básica de datos hasta funciones avanzadas de análisis y reporting. Su arquitectura flexible permite la adaptación a una variedad de necesidades, desde pequeñas aplicaciones hasta grandes empresas.

Esta base de datos relacional se destaca por su conformidad con los estándares SQL, lo que facilita la portabilidad y la interoperabilidad con otras soluciones de bases de datos. Además, PostgreSQL soporta la extensibilidad a través de la creación de funciones y la incorporación de extensiones, lo que permite a los desarrolladores personalizar y ampliar las capacidades de la base de datos según los requisitos específicos del proyecto.

Uno de los aspectos más notables de PostgreSQL es su compromiso con la integridad de los datos y la durabilidad de la información. La implementación de transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) asegura la fiabilidad de las operaciones, garantizando que la base de datos mantenga la coherencia incluso en situaciones de fallo del sistema.

En resumen, PostgreSQL emerge como una opción robusta y versátil para el diseño y desarrollo de bases de datos, ofreciendo un entorno confiable y eficiente para la gestión de datos en diversos contextos. Su comunidad activa, soporte continuo y capacidad para adaptarse a las necesidades cambiantes del mundo moderno hacen de PostgreSQL una elección sólida para aquellos que buscan una solución de base de datos potente y de código abierto.

4. CAPÍTULO III: METODOLOGÍA

4.1. Diseño metodológico

Para el desarrollo de este tema monográfico, que se define como una investigación de tipo aplicada, se realizó en diversas etapas, que incluyen la recolección de información sobre las plagas y enfermedades que afectan al cultivo de arroz en el departamento de León, las herramientas que utilizamos para manejo de datos y desarrollo de software... A continuación, mencionamos las etapas.

4.2. Etapas de la investigación

4.2.1. Etapa de exploración

En toda investigación, es fundamental la recolección de datos o información, en esta etapa realizamos un recorrido sobre diferentes parcelas de arroz, donde conversamos con los productores sobre las plagas y enfermedades más recurrentes, así como también las fechas que atacan. Por otra parte, visitamos diferentes organizaciones de gobierno que están relacionadas con la agricultura, también, obtuvimos información fundamental sobre tratamientos de cultivos en agro-comercios.

4.2.2. Etapa descriptiva

Luego del proceso de estudio pudimos observar en diferentes parcelas de arroz y conocer la opinión de los productores, notamos que un sistema para detectar diferentes enfermedades en el arroz sería muy útil para ellos.

4.2.3. Etapa proyectiva

En esta etapa se ideó una estrategia para darle solución y mejorar el control de las plagas y enfermedades en el cultivo de arroz a través de un sistema.

4.2.4. Etapa iterativa

La implementación de esta Inteligencia Artificial mediante una red neuronal convolucional para la detección de plagas y enfermedades en el cultivo de arroz sigue una etapa iterativa que busca mejorar progresivamente la precisión y eficacia del sistema. En esta fase, se recopilaron datos relevantes sobre las plagas y enfermedades que afectan al cultivo, así como imágenes de cultivos afectados.

Posteriormente, se dio la creación y entrenamiento del modelo de red neuronal convolucional, utilizando conjuntos de datos iniciales y refinando continuamente el algoritmo con base en la retroalimentación y los resultados obtenidos en cada iteración. Se realizaron pruebas exhaustivas del modelo en condiciones variables de cultivo y diferentes tipos de plagas y enfermedades, con el objetivo de identificar y corregir posibles deficiencias o errores. Esta iteración continua permite ajustar parámetros, mejorar la precisión del modelo y adaptarlo a las necesidades específicas de los agricultores, garantizando así un sistema robusto y efectivo para la detección temprana de amenazas en los cultivos de arroz.

4.2.4.1. Modelo de inteligencia artificial

Descripción general de la Red Neuronal Convolucional desarrollada para la detección de enfermedades en el arroz.

Se hará mediante la creación de un clasificador de enfermedades, que diga si una imagen contiene un tipo de enfermedad del arroz. Donde la red está entrenada para reconocer las características que determinan cuál es cuál. Se ha procesado una serie de imágenes, que servirá como el conjunto de entrenamiento para decirle al modelo de red neuronal que “este es un tipo específico de una enfermedad en el arroz”. No se realiza etiquetado de imágenes explícitamente, en su lugar, usamos ImageDataGenerator, que lee imágenes de subdirectorios y las etiquetas automáticamente a partir del nombre de ese subdirectorio. Por ejemplo, tenemos un directorio de entrenamiento que contiene un directorio Tizón Bacteriano y otro Carbón de hoja. ImageDataGenerator etiquetará las imágenes apropiadamente, esto nos ayuda a reducir un gran paso de codificación.

Definiendo el modelo

Comenzamos importando la biblioteca TensorFlow, esta nos facilitará la creación de modelos de aprendizaje automático. Luego, agregamos capas convolucionales y Flatten, que se utiliza para aplanar la entrada, por ejemplo, si se aplica Flatten a una capa que tiene una forma de entrada como (bacth_size, 3, 3), entonces la forma de salida de la capa será (batch_size, 9).

```
model = tf.keras.models.Sequential([
  # Note the input shape is the desired size of the image 300x300 with 3 bytes
  color
  # This is the first convolution
 tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 3)),

  tf.keras.layers.MaxPooling2D(2, 2),
  # The second convolution
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  # The third convolution
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  # The fourth convolution
  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  # The fifth convolution
  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Dropout(0.5),
  # Flatten the results to feed into a DNN
  tf.keras.layers.Flatten(),
  # 512 neuron hidden layer
  tf.keras.layers.Dense(512, activation='relu'),
  # Only 12 output neurons.
  tf.keras.layers.Dense(12, activation='softmax')

])
```

El modelo tiene una capa de entrada para una imagen 300 x 300 (altura y ancho, respectivamente), manejando 3 bytes de colores, ya que necesitamos saber todos los colores posibles al momento de predecir la enfermedad del arroz en la imagen. Esta capa es la que recibe primero la información del mundo exterior, donde los nodos de entrada procesan los datos y lo pasan a la siguiente capa.

Ahora, las capas ocultas, que consiste en 4 capas ocultas con activación relu, que se encargan de analizar la salida de la capa exterior donde la procesan aún más y la pasan la siguiente capa.

Por último, la capa de salida proporciona el resultado final de todo el procesamiento de datos que realiza la red neuronal artificial, en este caso, tenemos un problema de clasificación multiclase, la capa de salida puede estar formada por más de un nodo de salida (11 enfermedades que el modelo puede predecir).

Para lograr eficiencia en el modelo principal, nos dimos la tarea de crear un modelo para validar las imágenes enviadas del exterior, asegurándonos de que las imágenes son válidas para el modelo principal. En cuestión de código son similares, la diferencia se basa en que el modelo de validación solo nos da un resultado binario (1: válido - 0: no válido).

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 3)),
  tf.keras.layers.MaxPooling2D(2, 2),
  # The second convolution
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  # The third convolution
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  # The fourth convolution
  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  # The fifth convolution
  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Dropout(0.5),
  # Flatten the results to feed into a DNN
  tf.keras.layers.Flatten(),
  # 512 neuron hidden layer
  tf.keras.layers.Dense(512, activation='relu'),
  # Only 1 output neuron. binary_crossentropy
  tf.keras.layers.Dense(1, activation='sigmoid')

])
```

La llamada al método model.summary() imprime un resumen de la red:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 16)      448
 max_pooling2d (MaxPooling2D)  (None, 149, 149, 16)    0
 conv2d_1 (Conv2D)           (None, 147, 147, 32)      4640
 max_pooling2d_1 (MaxPooling2D)  (None, 73, 73, 32)    0
 conv2d_2 (Conv2D)           (None, 71, 71, 64)        18496
 max_pooling2d_2 (MaxPooling2D)  (None, 35, 35, 64)    0
 conv2d_3 (Conv2D)           (None, 33, 33, 64)        36928
 max_pooling2d_3 (MaxPooling2D)  (None, 16, 16, 64)    0
 conv2d_4 (Conv2D)           (None, 14, 14, 64)        36928
 max_pooling2d_4 (MaxPooling2D)  (None, 7, 7, 64)      0
 flatten (Flatten)           (None, 3136)              0
 dense (Dense)               (None, 512)               1606144
 dense_1 (Dense)             (None, 4)                 2052
=================================================================
Total params: 1705636 (6.51 MB)
Trainable params: 1705636 (6.51 MB)
Non-trainable params: 0 (0.00 Byte)
```

La columna Output Shape muestra cómo evoluciona el tamaño del mapa de características en cada capa sucesiva. Las capas de convolución reducen un poco el tamaño de los mapas de características debido al relleno y cada capa de agrupación reduce a la mitad las dimensiones.

Compilación del modelo

A continuación, configuramos las especificaciones para el entrenamiento del modelo. Es entrenado con la pérdida categorical_crossentropy porque es un problema de clasificación categórica, ya que intentamos predecir entre múltiples clases y la activación final es un softmax.

```
model.compile(loss=categorical_crossentropy,
        optimizer='adam',
        metrics=['acc']
        )
```

Entrenamiento del modelo a partir de generadores

Configuramos los generadores de datos que lean imágenes en nuestras carpetas de origen, las conviertan en tensores float32 y las alimente (con las etiquetas) a la red.

Tendremos un generador para las imágenes de entrenamiento y otro para las imágenes de validación. Los generadores producirán lotes de imágenes de validación de tamaño 300x300 y las etiquetas (binario).

Como ya sabemos, los datos que ingresan a las redes neuronales generalmente deben normalizarse de alguna manera para que sean más fáciles de procesar por la red. (Es poco común introducir píxeles sin procesar en una CNN). En nuestro caso, procesaremos las imágenes normalizando los valores de píxeles para que estén en el rango [0, 1], originalmente todos los valores están en el rango [0, 255].

En keras, eso se puede hacer a través de la clase keras.processing.image.ImageDataGenerator usando el parámetro rescale. Esta clase ImageDataGenerator permite crear instancias de generadores de lotes de imágenes aumentadas (y sus etiquetas) a través del método. flow(data, labels) o .flow_from_directory(directory). Estos generadores luego se pueden usar con los métodos del modelo Keras que aceptan generadores de datos como entradas: fit_generator, evaluation_generator y predict_generator.

```
# Preprocesando datos de entrenamiento
# Todas las imágenes serán reescaladas por 1./255
train_datagen = ImageDataGenerator(rescale=1 / 255,
                    shear_range=0.2,
                    zoom_range=0.2,
```

```
                    horizontal_flip=True
                    )
# Imágenes de entrenamiento de flujo en lotes de 128 usando el generador train_datagen
train_generator = train_datagen.flow_from_directory(
        utils.constant.all_states_training_dir(),  # Este es el directorio Fuente para las imágenes de entrenamiento.

        target_size=(300, 300),  # Todas las imágenes serán reescaladas a 300x300
        batch_size=129,
# Ya que usamos pérdida categorical_crossentropy, necesitamos etiquetas categorizadas.
        class_mode='categorical'
)
# Preprocesando datos de pruebas
test_datagen = ImageDataGenerator(rescale=1. / 255)
test_generator = test_datagen.flow_from_directory(
        utils.constant.all_states_test_dir(),
        target_size=(300, 300),
        batch_size=128,
        class_mode='categorical'
)
```

Entrenamos el modelo durante 100 épocas.

```
history = model.fit(
        x=train_generator,
        validation_data=test_generator,
        batch_size=32,
        epochs=100,
        validation_split=0.15,
        verbose=1)
```

Tengamos en cuenta los valores por época. La pérdida y la precisión son una gran indicación del progreso del entrenamiento. Se trata de adivinar la clasificación de los datos de entrenamiento y luego medirlos con la etiqueta conocida, calculando el resultado. La precisión es la porción de conjeturas correctas.

Para los datos de validación asignamos al generador de imágenes que hemos creado para el aumento de datos, los cuales se asemejan a las imágenes que podría predecir en la vida real.

De todos los datos de entrenamiento, solo utilizamos el 0.15% de datos para hacer las validaciones durante el entrenamiento. Eso se lo indicamos al parámetro validation_split.

Primeras 8 épocas

```
# 2/2 [==============================] - 18s 15s/step - loss: 0.5867 - acc: 0.2867 - val_loss: 0.6443 - val_acc: 0.2733
```

```
Epoch 2/100
2/2 [==============================] - 10s 3s/step - loss: 0.6289 - acc: 0.2733 - val_loss: 0.5793 - val_acc: 0.3067
Epoch 3/100
2/2 [==============================] - 10s 9s/step - loss: 0.5728 - acc: 0.3467 - val_loss: 0.5437 - val_acc: 0.2733
Epoch 4/100
2/2 [==============================] - 10s 9s/step - loss: 0.5354 - acc: 0.2733 - val_loss: 0.5046 - val_acc: 0.6333
Epoch 5/100
2/2 [==============================] - 10s 3s/step - loss: 0.4998 - acc: 0.6133 - val_loss: 0.4826 - val_acc: 0.4667
Epoch 6/100
2/2 [==============================] - 10s 3s/step - loss: 0.4781 - acc: 0.4667 - val_loss: 0.4908 - val_acc: 0.3867
Epoch 7/100
2/2 [==============================] - 10s 3s/step - loss: 0.4754 - acc: 0.4200 - val_loss: 0.4237 - val_acc: 0.5467
Epoch 8/100
2/2 [==============================] - 10s 3s/step - loss: 0.4217 - acc: 0.5400 - val_loss: 0.3892 - val_acc: 0.5400
Epoch 9/100
2/2 [==============================] - 10s 9s/step - loss: 0.4531 - acc: 0.4667 - val_loss: 0.3726 - val_acc: 0.6000
Epoch 10/100
2/2 [==============================] - 10s 9s/step - loss: 0.4547 - acc: 0.5200 - val_loss: 0.3934 - val_acc: 0.5600
Epoch 11/100
2/2 [==============================] - 9s 3s/step - loss: 0.3925 - acc: 0.5600 - val_loss: 0.3901 - val_acc: 0.5733
```

Últimas 10 épocas

```
Epoch 90/100
2/2 [==============================] - 9s 3s/step - loss: 0.1474 - acc: 0.8733 - val_loss: 0.0654 - val_acc: 0.9667
Epoch 91/100
2/2 [==============================] - 10s 3s/step - loss: 0.0712 - acc: 0.9533 - val_loss: 0.2106 - val_acc: 0.8133
Epoch 92/100
2/2 [==============================] - 10s 3s/step - loss: 0.1930 - acc: 0.8533 - val_loss: 0.1465 - val_acc: 0.8467
Epoch 93/100
2/2 [==============================] - 9s 3s/step - loss: 0.1669 - acc: 0.8467 - val_loss: 0.1199 - val_acc: 0.8867
Epoch 94/100
2/2 [==============================] - 9s 3s/step - loss: 0.1342 - acc: 0.8867 - val_loss: 0.1117 - val_acc: 0.9200
Epoch 95/100
2/2 [==============================] - 9s 8s/step - loss: 0.0958 - acc: 0.9400 - val_loss: 0.0861 - val_acc: 0.9133
Epoch 96/100
2/2 [==============================] - 9s 3s/step - loss: 0.0933 - acc: 0.9133 - val_loss: 0.1035 - val_acc: 0.8867
Epoch 97/100
2/2 [==============================] - 9s 3s/step - loss: 0.1063 - acc: 0.8933 - val_loss: 0.0752 - val_acc: 0.9733
Epoch 98/100
2/2 [==============================] - 9s 3s/step - loss: 0.0796 - acc: 0.9467 - val_loss: 0.0696 - val_acc: 0.9600
Epoch 99/100
2/2 [==============================] - 9s 3s/step - loss: 0.0824 - acc: 0.9533 - val_loss: 0.0744 - val_acc: 0.9400
Epoch 100/100
2/2 [==============================] - 8s 7s/step - loss: 0.0936 - acc: 0.9200 - val_loss: 0.0745 - val_acc: 0.9200
```

En resumen, durante las primeras épocas, la red estaba en una fase de ajuste y posiblemente mostraba señales de sobreajuste. Sin embargo, en las últimas épocas, se observa una mejora significativa, una buena generalización y una posible reducción del sobreajuste. Este comportamiento sugiere que el modelo ha aprendido de manera efectiva y ha alcanzado un estado más maduro de entrenamiento hacia el final.

4.2.4.2. Base de datos

La base de datos que se presenta constituye un componente fundamental en el marco de nuestro sistema de gestión logística. Diseñada con el propósito de almacenar, organizar y proporcionar acceso eficiente a datos clave relacionados con plagas y enfermedades, consejos de cultivos y la lógica necesaria para poder dar información dada una predicción.

A continuación, se hará énfasis a las entidades principales que conforman la base de datos:

Nombre de la entidad: User

Descripción: El usuario representa a individuos que interactúan con el sistema, accediendo y utilizando sus funcionalidades.

Atributos:

- id (primary key): int
- username: string
- password: string
- auth_strategy: string
- create_at: date

Relaciones:

- Prediction History: (Relación de uno a muchos): Relacionado con la entidad “Historial de predicciones” a través de user_id. Un Usuario puede tener varias predicciones.

Nombre de la entidad: Prediction

Descripción: Representa el código de clase y nombre de clase con la cual fue entrenado el modelo de inteligencia artificial.

Atributos:

- id (primary key): int
- class_code: string
- class_name: string

Relaciones:

- Prediction Result: (Relación de uno a uno): Relacionado con la entidad “Prediction Result” a través de prediction_id. Una predicción sólo puede tener un resultado con la información necesaria.

Nombre de la entidad: Prediction Result

Descripción: Representa la información necesaria de cada plaga o enfermedad que el modelo de inteligencia artificial pueda predecir.

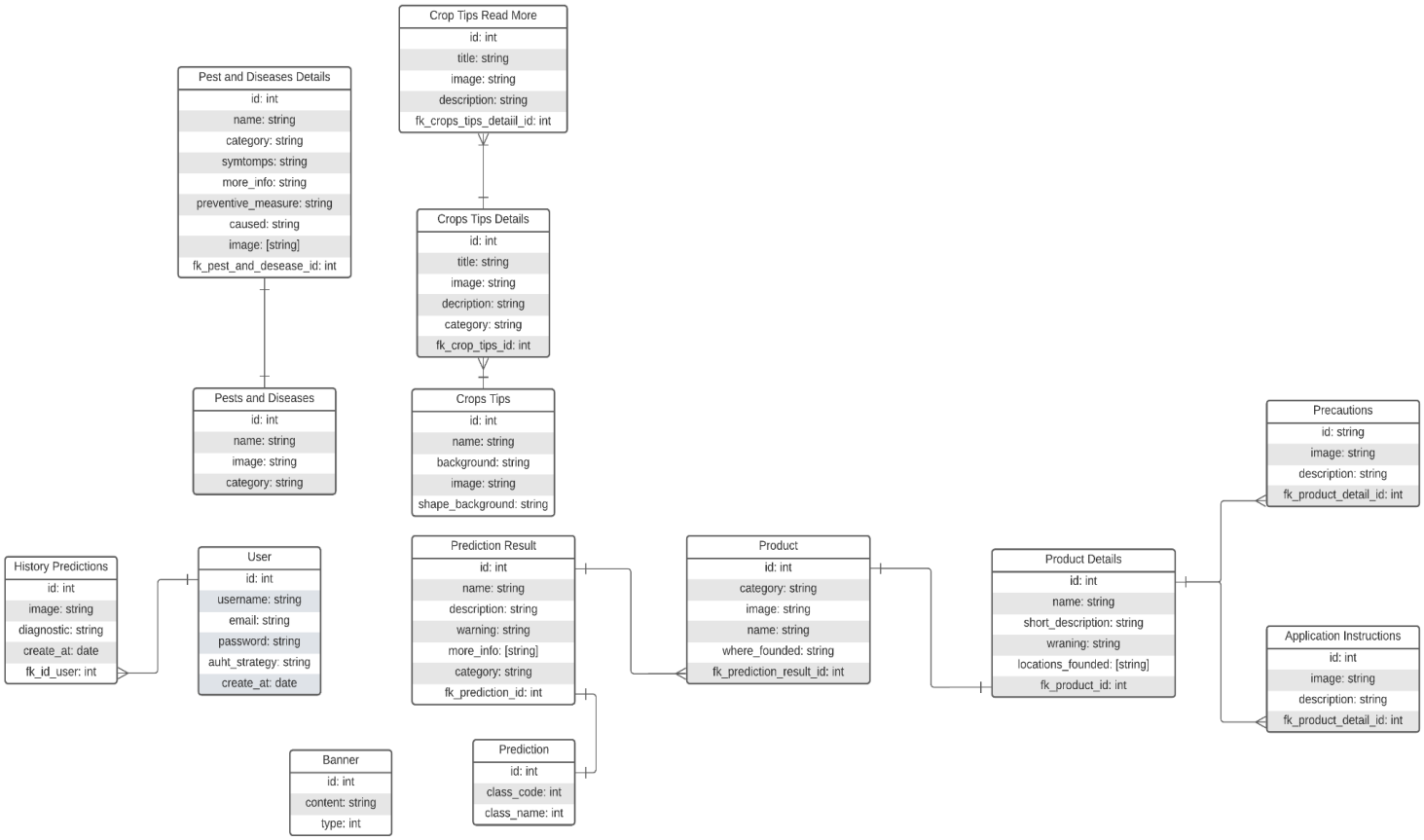
Atributos:

- id (primary key): int
- name: string
- description: string
- warning: string
- more_info: [string]
- category: string
- fk_prediction_id (foreign key): int

Relaciones:

- Prediction: (Relación de uno a uno): Relacionado con la entidad “Prediction”. Incluye una clave foránea que se asocia con una plaga o enfermedad específica que el modelo de inteligencia artificial tiene la capacidad de predecir.

Estas han sido algunas de las principales entidades que conforman la base de datos, para tener una visualización completa, se creó el diagrama entidad-relación que condensa información compleja de manera visual y estructurada, en el contexto de la base de datos utilizada para el desarrollo del sistema.



En conclusión, la implementación de esta base de datos se erige como un componente esencial para la gestión eficiente de la información en nuestro sistema. Al detallar la estructura de las entidades, relaciones y atributos.

La cuidadosa definición de objetivos específicos ha guiado cada aspecto de desarrollo, desde la creación de entidades claves como “User” hasta la representación detallada de “Predictions” y sus resultados asociados. La interconexión de estas entidades refleja la complejidad de nuestro sistema, permitiendo este acceso fluido a la información relevante y facilitando el análisis predictivo de nuestro modelo de inteligencia artificial.

4.2.4.3. Backend

En el complejo ecosistema de aplicaciones modernas, la integración de múltiples tecnologías se convierte en un pilar fundamental para alcanzar soluciones avanzadas y eficientes. En este contexto, la combinación estratégica de un backend potenciado por FastAPI, reconocido por su agilidad y rendimiento, con un modelo de inteligencia artificial encapsulado, despliega una infraestructura sólida para el procesamiento de datos complejos.

Este backend en FastAPI no solo sirve como una interfaz robusta para recibir y procesar solicitudes, sino que también actúa como un repositorio para un modelo de inteligencia artificial. Cuando recibe una solicitud proveniente de un backend en Nest.js, este backend en FastAPI es capaz de recibir imágenes, realizar un análisis profundo mediante el modelo de inteligencia artificial y devolver los resultados procesados.

Una vez que el modelo de inteligencia artificial ha procesado la imagen y ha generado predicciones, el backend de FastAPI retorna la información procesada al backend en Nest.js. Aquí, el backend de Nest.js utiliza estos datos para realizar consultas a una base de datos, buscando coincidencias o información relevante con respecto a las predicciones generadas por el modelo de IA.

Esta integración fluida entre los backends permite una interacción dinámica y eficiente entre la capacidad predictiva del modelo de inteligencia artificial, el procesamiento de datos realizado por el backend de Nest.js y el almacenamiento y recuperación de información relevante desde la base de datos. Esta colaboración facilita la generación de respuestas precisas y contextualizadas en función de las predicciones de la IA, convirtiéndose en una base sólida para aplicaciones avanzadas y centradas en el análisis de datos y la inteligencia artificial en tiempo real.

FastApi

A continuación, se describe el módulo principal que hace posible la predicción de imágenes, haciendo uso del modelo pre entrenado.

predicction_controller.py

En este archivo podemos entrenar la función que se encarga de recibir y normalizar la imagen para que sea válida para la capa de entrada.

```
ImageFile.LOAD_TRUNCATED_IMAGES = True
resource_response = requests.get(prediction.url_image)
image_file = Image.open(io.BytesIO(resource_response.content))
pil_image = image_file.resize((300, 300))
```

Como se puede notar, hacemos el rescate de la imagen a 300 x 300, ya que la capa de entrada del modelo sólo permite estas dimensiones.

Antes de enviar la imagen al modelo, hacemos la validación de la imagen con el modelo de validación.

```
is_valid_image = make_image_validation(prediction_array)
```

Una vez que sabemos que la imagen es válida para el modelo principal, se la enviamos y hacemos la predicción, al mismo tiempo, obtenemos el índice con mayor probabilidad entre 0 y 1 de todas las clases posibles.

```
model = tf.keras.models.load_model(source_dir, compile=True)
classes = model.predict(prediction_array)[0]
likely_class = np.argmax(classes)
```

Veremos una salida como la siguiente, donde el índice 9 es nuestro resultado:

[1.263e-11,4.1e-13,0,4.14425e-9,2.8e-13,0,0.00000474586477,2.01658e-9,0,0.99999523162842,2.4e-13]

NestJS

prediction.controller.ts

Para el caso de NestJS, el controlador tiene una comunicación directa con la base de datos. Aquí se define la ruta que se encarga de recibir la imagen y almacenarla en el mismo, creando una dirección url para la imagen, esta url es enviada al backend de FastApi.

```
@Post(':device_id')
@UseInterceptors(
FileInterceptor('file', {
  storage: diskStorage({
    destination: 'public',
    filename: (req, file, cb) => {
      const fileName = file.originalname.replace(/\s/g, "_");
      FileName.fileName = fileName
      cb(null, fileName)
    },
  }),
}))
```

Una vez recibida la imagen y almacenada en el backend, podemos enviarle la url a través del servicio.

prediction.service.ts

El objetivo de este servicio es enviarle la solicitud al backend de FastApi para que analice la imagen, una vez responde FastApi, el servicio debería de tener dos posibles respuestas, la imagen no es válida o la imagen fue válida y ha sido analizada, retornando las predicciones y la clase con mayor probabilidad.

```
const { data } = await firstValueFrom(
  this.httpService.post<PredictionDTO>(API_IA_URL,
  createPredictionDto).pipe(
    catchError((err: AxiosError) => {
      throw new HttpException(`An occurred when trying get prediction from IA model: ${err}`, 500)
    })
  )
)
```

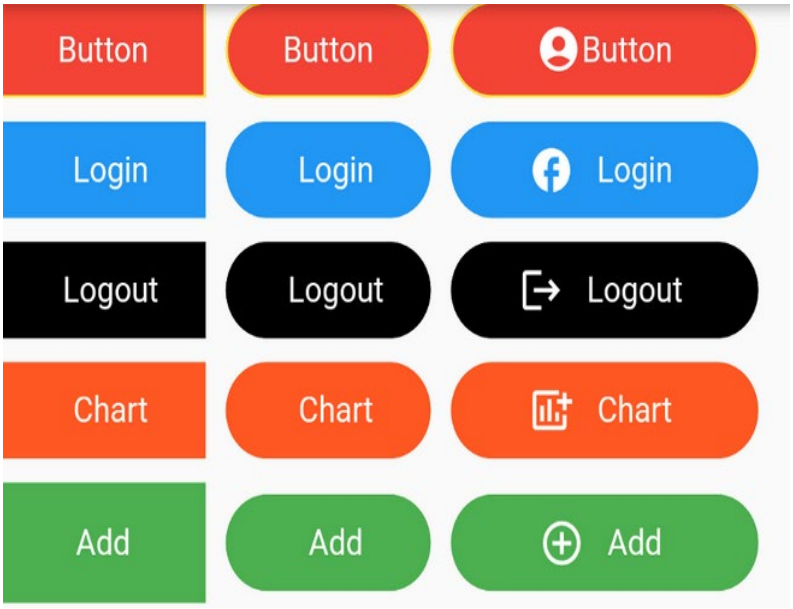
Teniendo los resultados de la predicción, podemos buscar en la base de datos la información basada en la clase con mayor probabilidad, asimismo, guardando el resultado en el historial del usuario.

```
const predictionResult = await this.predictionRepository.findOne({
  where: {
  id: data.likely_class
 },
 relations: ['products']
})
await this.saveHistoryPrediction(
 deviceId, {
  image: predictionResult.images,
  diagnostic: predictionResult.description
 })
```

4.2.4.4. Aplicación móvil

Durante el desarrollo de nuestra aplicación móvil como parte entregable para el usuario final hemos decidido utilizar Flutter como herramienta para tener nuestro código multiplataforma ya que nos brinda una gran ayuda en consideración que al estar haciendo desarrollo nativo y así estar preparados para un despliegue en múltiples plataformas tales como App Store, Gallery App y por supuesto nuestros propios medios.

Siendo Dart el lenguaje que utiliza Flutter como código fuente para generar maravillosas aplicaciones y al tener una manera de gestionar nuestro código de tal manera que sea tan reutilizable como sea posible. Empezando así con la separación tanto así que en Flutter todo es considerado un Widget, ¿Has visto un Botton en Flutter?



Esto es un Widget en flutter, absolutamente todo es un widget, y esto no es todo, hay dos clases bases de las cuales puede extender un Widget, estas son: Stateful y Stateelss widgets lo que diferencia a uno del otro es su rendimiento en el stateless y el conocimiento de en qué estado se encuentra el stateful widget , esto puede ser muy útil cuando necesitamos hacer dispose de algunos métodos, por ejemplo cuando usamos un WebSocket, no queremos que nuestra pantalla esté consumiendo recursos mientras no estemos en cierto punto de nuestra app, por ejemplo en una app donde tengamos una pantalla llamada “Rutas” o “Mapas”, en donde podemos ver nuestro transcurso en tiempo real, pero nos vamos a ajustes, necesitamos hacer un dispose del controlador o un kill al render de esa vista.

En cambio, con los stateless tenemos un rendimiento mayor ya que no están al pendiente esos recursos pueden ser ocupados por un proceso con mayor

prioridad. También pero no menos importante hablar del manejo de estados ya que Flutter cuenta con distintos paquetes para manejar estados también así el gestor que está en el núcleo del framework, de los cuales están los más populares: Bloc, Provider, GetX, y por supuesto el poco usado, pero ya incorporado Inherited Widget.

4.3. Etapa de pruebas de la aplicación

En esta etapa del proyecto, realizamos pruebas en entornos reales, en dispositivos físicos y parcelas de arroz con algún tipo de plaga.

4.4. Etapa de documentación

En esta última etapa, se procedió a la documentación de todo lo desarrollado durante este período. Donde se redactan los casos de uso y resultados finales, con el objetivo de llegar a conclusiones para el informe final.

4.4.1. Selección de herramientas

A continuación, mencionaremos las herramientas, lenguajes de programación y materiales hardware utilizados para llevar a cabo este proyecto, todo fue utilizado en Portátil MSI con 24GB de RAM, procesador Core I5 de 11° generación, tarjeta gráfica GTX 1150, 1 TB M2 de almacenamiento.

4.4.1.1. Herramientas de desarrollo

Nombre	Descripción	Costo
Visual Studio Code	Editor de código para compilar y depurar aplicaciones web y móviles.	Ninguno
Android Studio	Entorno de Desarrollo para aplicaciones Android.	Ninguno
DataGrip	Entorno de Desarrollo para manejo de base de datos y SQL.	Ninguno
Anaconda	Es una distribución de Python con paquetes relacionados con la ciencia de datos.	Ninguno
PyCharm	Entorno de Desarrollo para compilar programas en Python.	Ninguno
Spyder	Entorno de Desarrollo para programación científica en Python.	Ninguno
Lucidchart	Web utilizada para creación de modelos y diagramas de flujos.	Ninguno
Jira	Software de administración de tareas.	Ninguno
Github	Sistema de control de versiones de proyectos de software	Ninguno
WebStorm	Entorno de desarrollo del paquete de JetBrains para JavaScript y tecnologías relacionadas.	

4.4.1.2. Lenguajes de programación y librerías

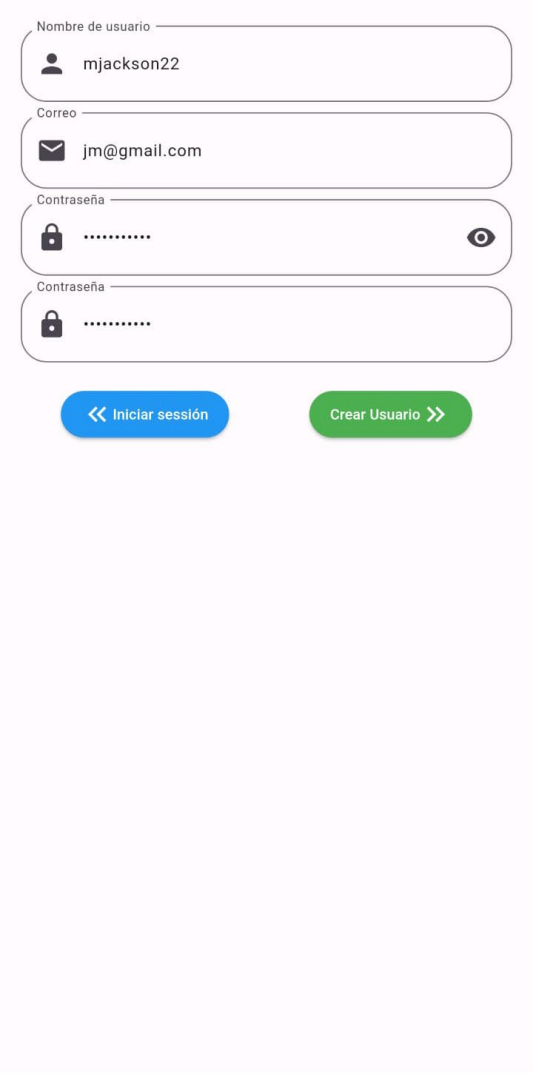
Nombre	Descripción	Costo
Python	Utilizado para el desarrollo de la Inteligencia Artificial.	Ninguno
Flutter	Utilizado para el desarrollo de la aplicación móvil multiplataforma Android/iOS.	Ninguno
TypeScript	Utilizado para el desarrollo del servidor que alimentará a la aplicación móvil.	Ninguno
Flask	Framework escrito en Python para almacenar el modelo entrenado.	Ninguno
PostgreSQL	Sistema de gestión de base de datos relacionales.	Ninguno
Express	Se utiliza para desarrollar aplicaciones web y APIs	Ninguno
Tensor Flow	Biblioteca para construir y entrenar redes neuronales	Ninguno

5. Casos de uso

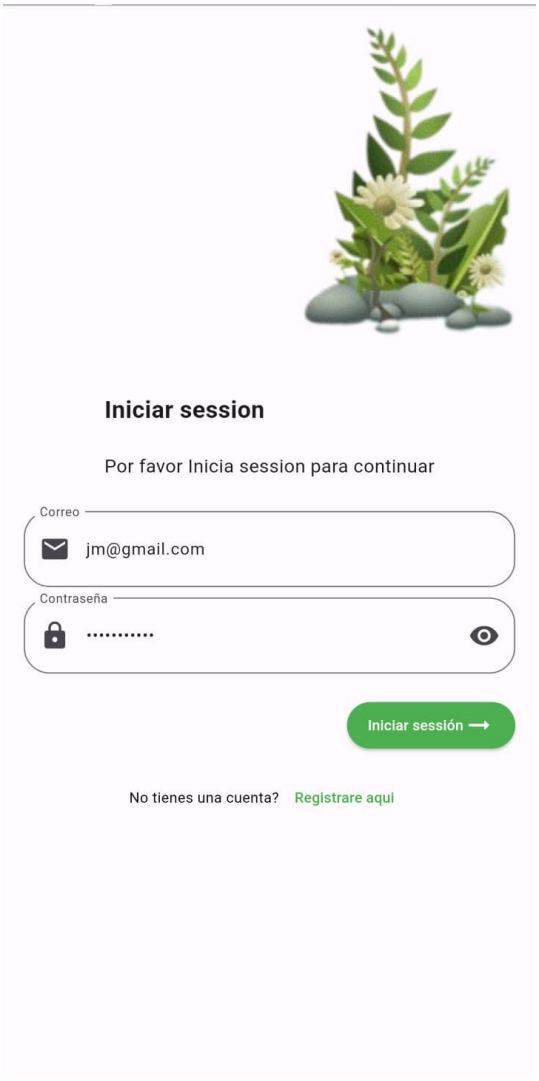
5.1. Interfaces

5.1.1. Autenticación

Para ser capaz de entrar a la aplicación es necesario que el usuario tenga que crear una cuenta y después iniciar sesión, esto para mantener un historial relacionado de las predicciones realizadas.



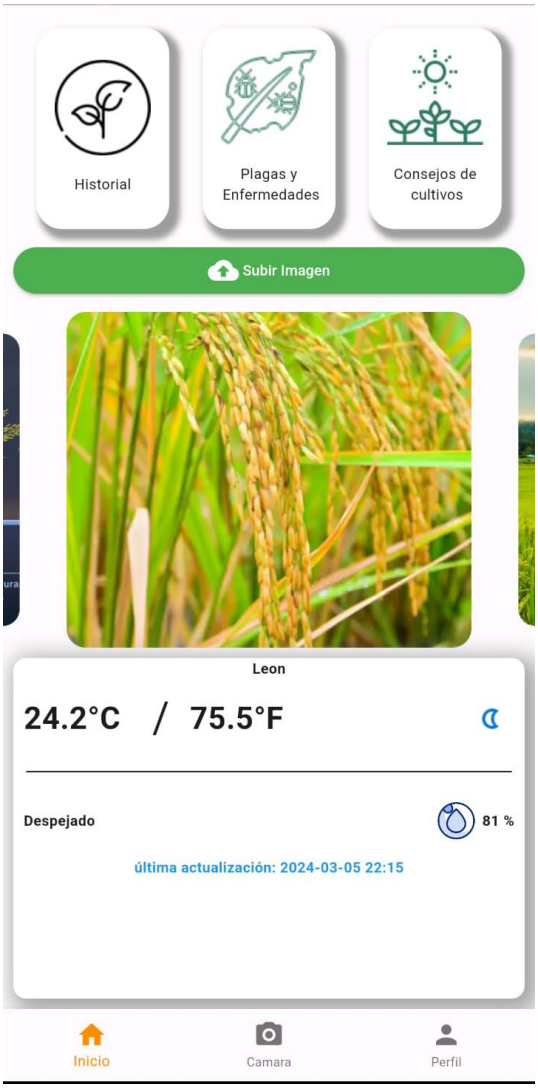
Crear usuario



Iniciar sesión

5.1.2. Pantalla Inicio

La pantalla inicio viene después de haber iniciado sesión, donde podemos tener una visión general para la navegación a otros módulos (Historial, Plagas y enfermedades, Consejos del cultivo, etc.), además información sobre el clima actual.



Pantalla de inicio

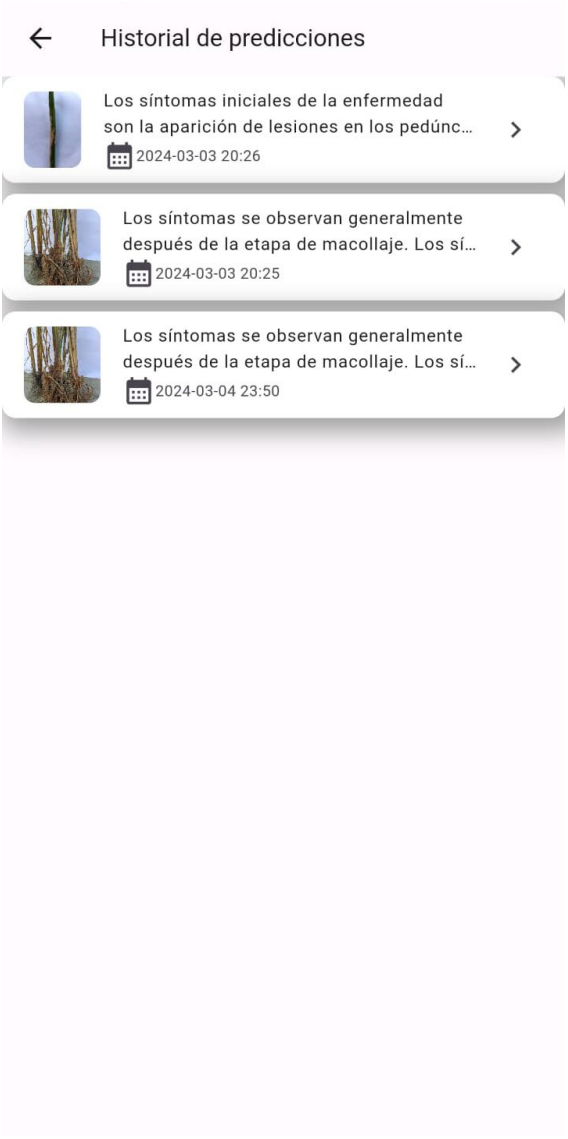
5.1.3. Diagnóstico

Al tomar una foto de la planta de arroz nos lleva a la pantalla de predicción donde nos muestra el tipo de enfermedad con descripción de los síntomas, también nos da algunas indicaciones de qué productos podemos utilizar.

Predicción

5.1.4. Historial

La pantalla de Historial guarda un registro de todas las predicciones que el usuario ha realizado, de este modo, permitimos al usuario tener acceso rápido a la información de enfermedades previamente analizadas. Al presionar un elemento, esta nos llevará a los detalles de ese historial, mostrando el resultado de su respectivo diagnóstico.

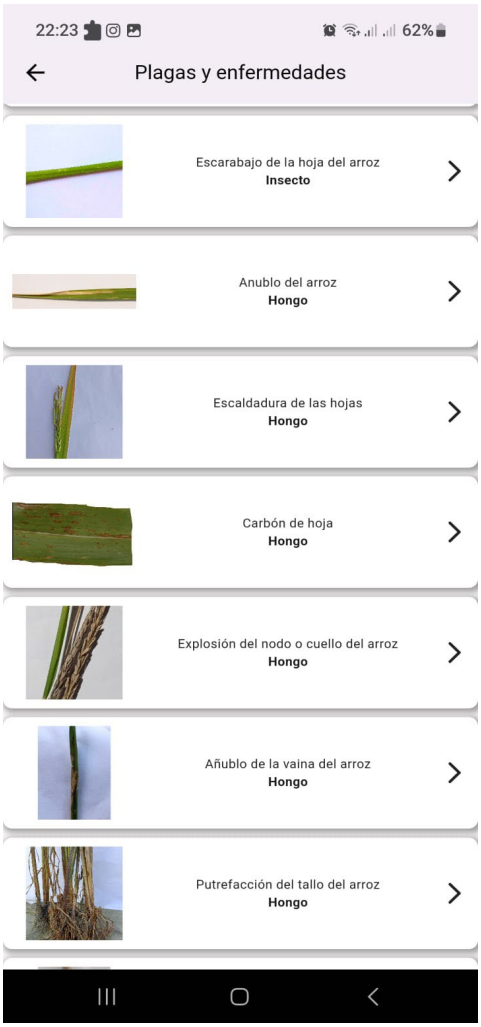


Historial de diagnósticos

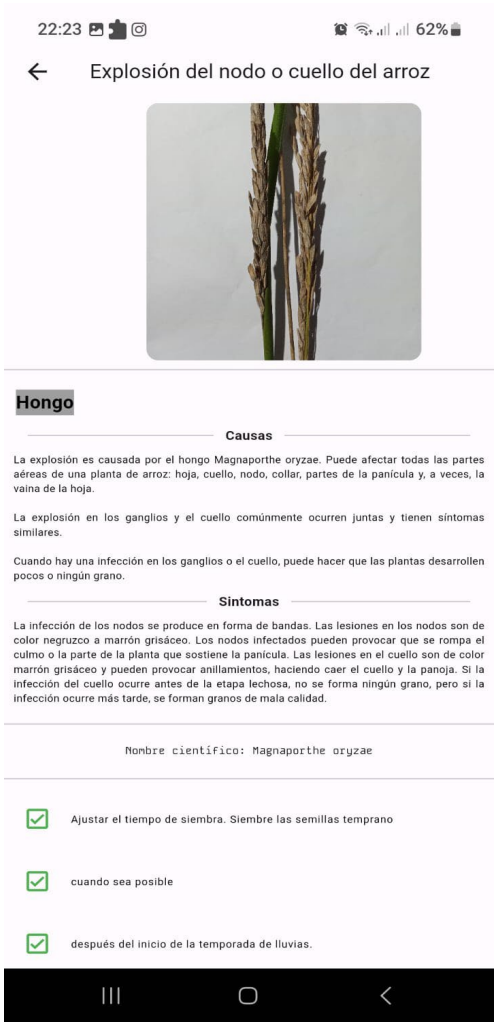
Detalles del historial

5.1.5. Plagas y enfermedades

La pantalla de plagas y enfermedades no muestra varias afectaciones que suelen presentar los cultivos de arroz al presionar una de estas, no muestra otra pantalla con las posibles causas de enfermedad y sus síntomas.



Plagas y enfermedades

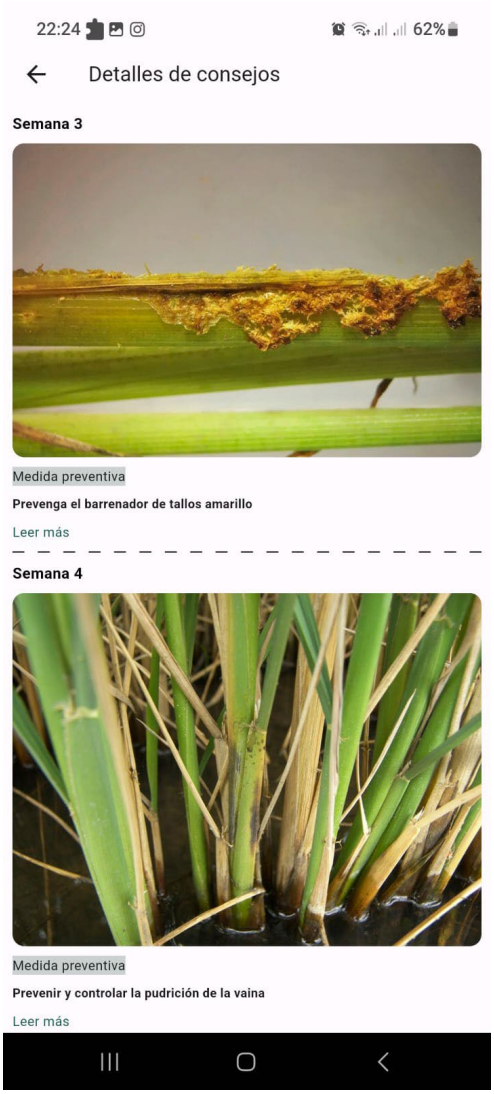


Detalles de Plagas y enfermedades

5.1.6. Consejos del cultivo

En la siguiente pantalla se muestran consejos enfocados a mantener un buen cuidado de la siembra, así como el fortalecimiento de esta. Al presionar uno nos da los detalles de consejos.

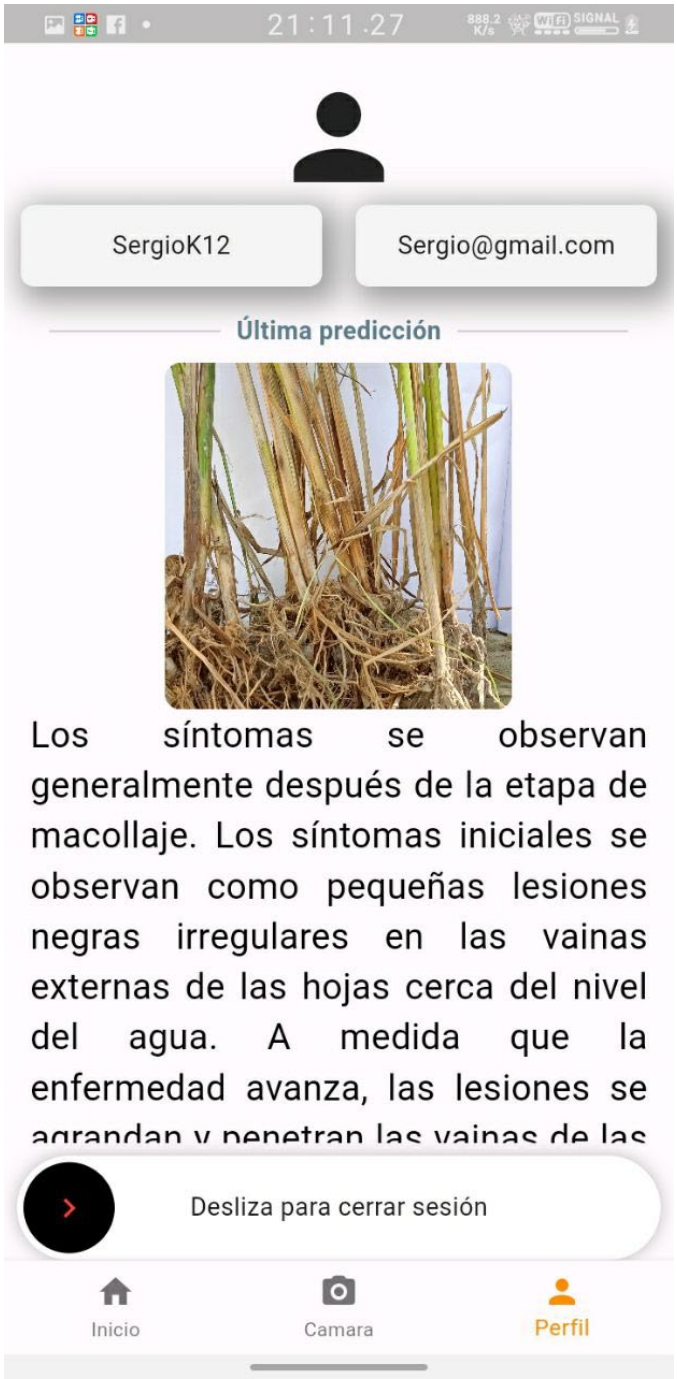
Consejos de cultivos



Detalles de consejos de cultivos

5.1.7. Perfil

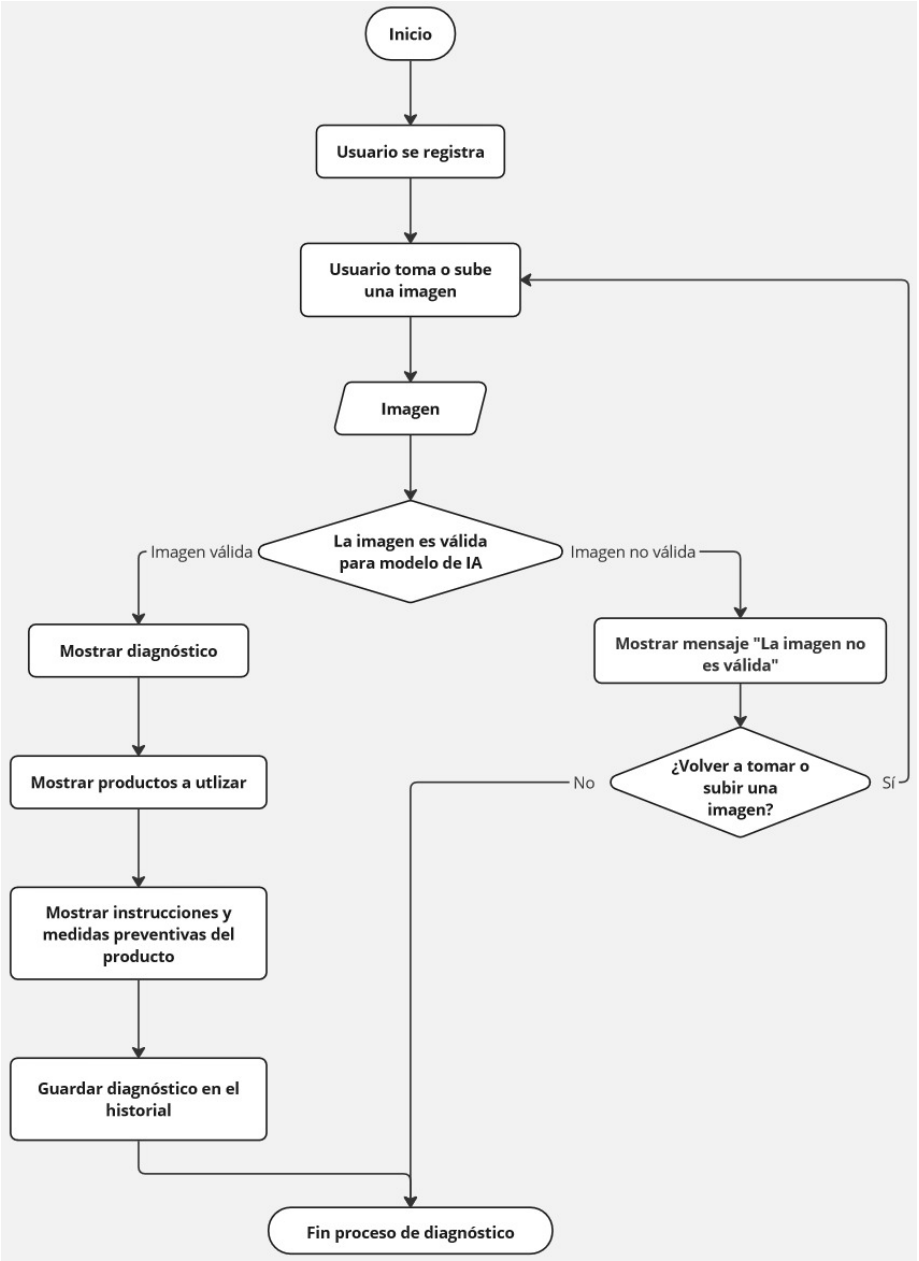
En esta pantalla muestra información relacionada al usuario, como su nombre de usuario y correo, además de la última predicción que ha realizado.



Perfil

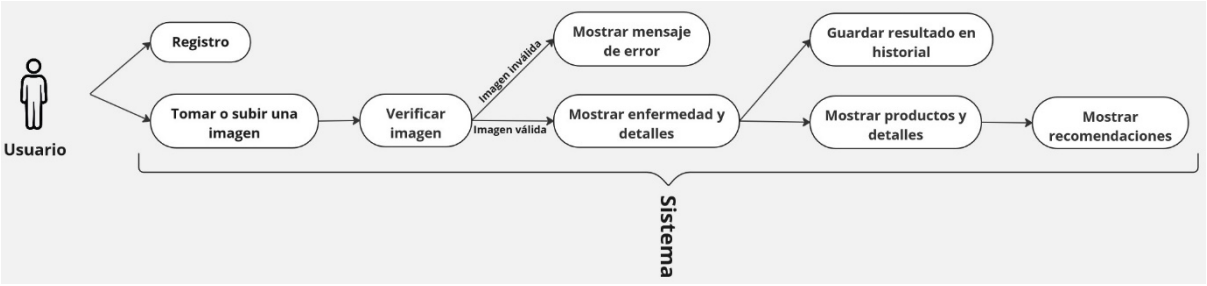
5.1.8. Diagrama de Flujo

El siguiente diagrama de flujo describe el proceso en el que un usuario se registra y luego toma o sube una imagen. Esta imagen es verificada en una base de datos para determinar su validez. Si la imagen es válida, se muestran detalles sobre la enfermedad detectada, así como productos recomendados para tratarla, junto con sus detalles y recomendaciones. Posteriormente, la predicción se guarda en el historial del usuario. Si la imagen no es válida, se muestra un mensaje correspondiente y se ofrece la opción de reintentar. Una vez que el proceso se completa, el flujo se detiene.



5.1.9. Diagrama de Caso de Uso

Este caso de uso ilustra el flujo principal de interacción entre el usuario y el sistema para el análisis y tratamiento de enfermedades basado en una imagen proporcionada por el usuario. Inicia con el registro del usuario en el sistema y luego la captura o carga de una imagen. Si la imagen es válida, se verifica su correspondencia con la base de datos de imágenes almacenadas. Si hay una coincidencia, se muestran detalles sobre la enfermedad detectada, junto con los productos recomendados y sus detalles para el tratamiento, así como recomendaciones adicionales. Posteriormente, la predicción se guarda en el historial del usuario. En caso de que la imagen no sea válida o no coincida con la base de datos, se muestra un mensaje apropiado. Una vez que se completa el proceso, el flujo se detiene.



6. Conclusión

En conclusión, el proyecto cumple con el objetivo principal de un sistema de integral para la detección de plagas y enfermedades en el cultivo de arroz mediante el uso de redes neuronales convolucionales.

Con los requerimientos previamente establecidos pudimos realizar las bases con las cuales nos guiamos para desarrollar el sistema.

De igual modo con la investigación y recopilación de datos permitieron identificar las funcionalidades esperadas, guiando así el proceso de desarrollo del modelo de Inteligencia Artificial especializado.

Al tener los puntos claves de nuestra investigación fue momento de encontrar un modo por el cual nos sea más fácil controlar la información que obtuvimos, es de este modo que se construyó una aplicación back-end para la gestión y manejo de la información necesaria para el sistema

Como punto final, el modo de que el usuario promedio pueda utilizar esta tecnología se construyó la aplicación móvil que consolida todo lo anterior mencionado, en una interfaz fácil de usar.

7. Recomendaciones

Validación exhaustiva del modelo: Antes de implementar el sistema en un entorno operativo, es crucial realizar una validación exhaustiva del modelo de Inteligencia Artificial. Esto implica evaluar su rendimiento con conjuntos de datos variados y representativos, asegurándose de que la red neuronal convolucional sea capaz de detectar plagas y enfermedades de manera precisa y consistente.

Actualización continua del modelo: La agricultura es un campo dinámico, con la evolución constante de nuevas cepas de plagas y enfermedades. Por lo tanto, se recomienda establecer un proceso para la actualización regular del modelo de IA. La retroalimentación de los usuarios y la inclusión de nuevos datos en el entrenamiento son esenciales para mantener la efectividad del sistema a lo largo del tiempo.

Integración con sistemas de información existentes: Para mejorar la utilidad y la aceptación del sistema, es importante considerar la integración con otros sistemas de información agrícola existentes. Esto facilita la interoperabilidad y permitiría a los agricultores acceder a datos relevantes de manera centralizada.

Capacitación y soporte técnico: Proporcionar capacitación adecuada a los usuarios, como agricultores y técnicos agrícolas, es esencial para asegurar una adopción exitosa. Además, se debe establecer un sistema de soporte técnico eficiente para abordar cualquier problema o pregunta que puedan tener los usuarios durante la implementación y el uso continuo del sistema también utilizar un sistema de calificación, donde el usuario pueda puntuar su satisfacción al igual dejar comentarios para posibles mejoras.

Consideraciones de privacidad y seguridad: Dado que el sistema implica la recopilación y gestión de datos agrícolas, es fundamental abordar las preocupaciones de privacidad y seguridad. Implementar medidas robustas de protección de datos y garantizar la conformidad con las regulaciones de privacidad agrícola será crucial para ganar la confianza de los usuarios.

Evaluación del impacto ambiental: Evaluar el impacto ambiental del sistema es importante, especialmente si se espera que tenga un despliegue a gran escala.

Esto incluye considerar cómo la implementación del sistema puede influir en las prácticas agrícolas y si hay oportunidades para reducir el uso de productos químicos a través de una detección temprana y precisa de plagas y enfermedades.

Colaboración con expertos agrícolas: Proporcionar los contactos directos para la colaboración con expertos agrícolas, investigadores y extensionistas que puedan enriquecer el desarrollo del sistema. Su experiencia práctica puede proporcionar información valiosa sobre las plagas y enfermedades específicas del cultivo de arroz y contribuir a la mejora continua del sistema.

Al abordar estas recomendaciones, el sistema de detección de plagas y enfermedades en el cultivo de arroz puede ser más efectivo, adaptable y sostenible a largo plazo.

8. Bibliografía

Javier, F. (2021). Tipo de Redes Neuronales. Inteligencia-Artificial.dev. [Enlace] https://inteligencia-artificial.dev/tipos-redes-neuronales/

International Business Machines. (s. f.). ¿Qué son las redes neuronales convolucionales? IBM. Recuperado el 1 de noviembre de 2023, de https://www.ibm.com/es-es/topics/convolutional-neural-networks

Introducción a la arquitectura Flutter. (4 de febrero de 2022). Bigcode. [Enlace] https://bigcode.es/introduccion-a-la-arquitectura-de-aplicaciones-con-flutter/

Flutter. (s. f.). Esflutter.dev. [Enlace] https://esflutter.dev/

Manual de Aprendizaje QGIS. (s. f.). Docs.qgis. Recuperado el 28 de noviembre de 2023, de https://docs.qgis.org/3.28/es/docs/training_manual/database_concepts/index.html

Gomez, M. (5 de abril de 2021). Inteligencia artificial y desarrollo móvil. Encora.com. [Enlace] https://www.encora.com/es/blog/inteligencia-artificial-y-desarrollo-movil

La inteligencia artificial ahora puede estimar el rendimiento del arroz. (s. f.). Mundo Agropecuario. Recuperado el 28 de noviembre de 2023, de https://mundoagropecuario.com/la-inteligencia-artificial-ahora-puede-estimar-el-rendimiento-del-arroz-segun-un-nuevo-estudio/

Spyder, un potente entorno de desarrollo interactivo para Python. (s. f.). Ubunlog. [Enlace] https://ubunlog.com/spyder-entorno-desarrollo-python/

Entorno de desarrollo Anaconda. (s. f.). Data Scientist en Python - Nivel Básico. [Enlace] https://entrenamiento-data-scientist-python.readthedocs.io/leccion1/anaconda.html

Android Developers. (7 de agosto de 2023). Introducción a Android Studio. Developers. Recuperado el 28 de noviembre de 2023, de https://developer.android.com/studio/intro?hl=es-419

9. Anexos

Describe el flujo del funcionamiento del sistema