UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE GRADO PREVIO A LA OBTENCION DEL TÍTULO DE INGENIERO EN MECATRÓNICA

TEMA:

"CLASIFICACIÓN DE PLÁNTULAS POR VISIÓN ARTIFICIAL"

AUTOR: JHENIFER DAYANA GUAGALA ANDRANGO

DIRECTOR: CARLOS XAVIER ROSERO CHANDI

IBARRA - ECUADOR

JULIO 2022

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

1. IDENTIFICACIÓN DE LA OBRA

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

DATOS DEL AUTOR	
Cédula de identidad:	1004391775
Apellidos y nombres:	Jhenifer Dayana Guagala Andrango
Dirección:	Babahoyo y Pelícano – Ibarra
Email:	jdguagalaa@utn.edu.ec
Teléfono móvil:	0999172301
DATOS DE LA OBRA	
Título:	"Clasificación de Plántulas por Visión Artificial"
Autor:	Jhenifer Dayana Guagala Andrango
Fecha:	6 de julio del 2022
Programa:	Pregrado
Título por el que opta:	Ingeniero en Mecatrónica
Asesor/director:	Carlos Xavier Rosero Chandi

2. CONSTANCIAS

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

Ibarra, a los 06 días del mes de julio de 2022.

EL AUTOR:

Jhenifer Dayana Guagala Andrango

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CERTIFICACIÓN

En calidad de director del trabajo de grado "CLASIFICACIÓN DE PLÁNTULAS POR VISIÓN ARTIFICIAL", presentado por la egresada JHENIFER DAYANA GUAGALA ANDRANGO, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, a los 06 días del mes de julio de 2022.

Firmado electrónicamente por:
CARLOS XAVIER ROSERO CHANDI

Carlos Xavier Rosero Chandi
DIRECTOR DE TESIS

Agradecimiento

El trabajo para la elaboración de la presente investigación no inició hace poco, sino que fue el último peldaño de 5 años de carrera en la cual pude vivir experiencias que han transformado mi forma de ser y han aumentado mi nivel de conocimiento, esto gracias a grandes personas como mis compañeros y mis maestros, por lo que estaré siempre muy agradecida.

Un complemento muy importante para mí son aquellas personas que están fuera de mi vida universitaria, pues, aunque no lo hayan notado todos ustedes impactaron directa e indirectamente en mi camino de cumplir una de mis metas más importantes hasta ahora.

Quiero agradecer de manera especial a mi director de tesis, Xavier Rosero, por encaminarme con sus conocimientos, pero principalmente por ser un ejemplo de humildad, paciencia y fuerza. Finalmente, gracias a mis amigos e integrantes del grupo de trabajo de grado por apoyarnos siempre en esta última etapa de la carrera.

Jhenifer Guagala Andrango

Dedicatoria

Este trabajo está dedicado enteramente a mis padres, Inés y Luis, pues fue por ellos que pude ser la persona que soy y que me gusta ser, es por ellos que siempre tengo la motivación para llegar a cada una de mis metas, su amor, su dedicación y sus palabras oportunas fueron claves para que hoy yo esté cumpliendo un sueño que no solo es mío, sino de ellos también.

Jhenifer Guagala Andrango

Resumen

El presente documento se muestra el proceso para desarrollar un algoritmo capaz de identificar hasta 4 tipos de plántulas, las cuales son: pamplina, maíz, bolsa de pastor y remolacha azucarera. Para cumplir dicho objetivo se inicia con la búsqueda de una base de datos en la plataforma Kaggle, ésta debe contener la información necesaria de las 4 plántulas ya mencionados. En la misma plataforma también existe un trabajo que utiliza la base de datos seleccionada para entrenar una Red Neuronal Convolucional (CNN) que identifica 12 tipos de plántulas. Desde este punto se trabaja en la modificación de la CNN para que trabaje solo con las 4 plántulas de interés, alcanzando una precisión del 98,95 %. A continuación, se realizan pruebas que evalúan la capacidad de reconocimiento de la CNN utilizando fotografías nuevas y otras que, ya usadas durante el entrenamiento, ambos grupos pertenecientes a la misma base de datos. Los resultados son altamente satisfactorios por lo que se guarda el modelo para ser utilizado como base para el desarrollo de la etapa final del algoritmo, el detector de objetos, en el que se hace uso de pirámides de imagen, ventanas deslizantes y supresión no máxima. Al aplicar estas técnicas se crean dos ventanas, en una se muestra la fotografía original y en la otra se muestra un recuadro verde rodeando al objeto a detectar junto a una etiqueta de la clase de plántula a la que representa. Finalmente, el algoritmo se completa con la conexión a una cámara que capture la imagen a analizar, para esto se emplea la aplicación DroidCam que permite la comunicación entre la computadora y un celular, siendo este último el encargado de tomar fotografías. Su porcentaje de precisión es del 86,67 %, un valor que puede aumentar al mejorar la calidad de las imágenes y que a su vez respeten el entorno de funcionamiento del algoritmo.

Abstract

This project starts developing an algorithm capable of identifying up to 4 types of seedlings: chickweed, maize, shepherds purse and sugar beet. The basis of the algorithm is a free use database available on the Kaggle platform, in which, casually, there is also a previous work that uses this database to train a Convolutional Neural Network (CNN) with favorable results, recognizing between 12 types of seedlings. With the database and CNN selected, the code of the latter was modified to reduce the number of classes to identify with the four already mentioned. The new CNN had an accuracy of 98,95 %, a valid percentage to start the tests using photographs that participated and not in the training, both groups belonging to the database, these results were enough to save the model that was used as a basis for the development of the final phase of the algorithm, the object detector, in which the techniques of image pyramids, sliding windows and non-maximum suppression were used, to show two windows, one with the original photograph and the other with a green box surrounding the object to be detected, as well as a label of the kind of seedling it represents. Finally, an extra program is included that using the DroidCam application that allows the communication among the computer with the camera on a cell phone to take photographs to analyze. Its percentage of precision was 86,67 %, a value that can increase as the sample number increases, always respecting the work environment.

Índice general

Índice de figuras

Índice de cuadros

Introducción

Este trabajo de grado ha sido realizado con el Grupo de Investigación en Sistemas Industriales y biomecánica de la Universidad Técnica del Norte (SIBI-UTN).

Objetivos

Objetivo general

Desarrollar un método para la clasificación de plántulas empleando visión artificial dentro del proceso del cultivo hidropónico.

Objetivos específicos

- Extraer las características de las hojas de plántulas útiles para el reconocimiento e interpretación de datos.
- Resolver el problema de clasificación utilizando técnicas de aprendizaje de máquina.
- Validar el método propuesto.

Problema

En el Ecuador, la agricultura es uno de los ejes principales de la economía, a finales del 2018 representó un 8 del Producto Interno Bruto (PIB) y durante los últimos años se ha trabajado por incrementar la productividad y así cumplir con la demanda interna y externa. Solo en Imbabura se cuenta con 283.659 hectáreas destinadas al cultivo, según el informe entregado por el Instituto Nacional de Estadística y Censo (INEC) en el 2018.

En el afán de incrementar el número de productores, el gobierno ha iniciado campañas estratégicas según las condiciones de cada provincia, asegurando precios de venta, entregando semillas certificadas, ofreciendo asesoramiento a los agricultores, implementando la tecnología con proyectos como el "Smart Farming" o Agricultura Inteligente que busca minimizar los desperdicios y maximizar la productividad en el mismo terreno de cultivo. Y de forma independiente aparece una alternativa más compleja como son los cultivos hidropónicos que dan un giro total a la agricultura tradicional ya que cambian el suelo por agua, consiguiendo cultivos más limpios y saludables que los cultivos en tierra.

Con respecto a este último, es importante conocer que el sistema no está diseñado para facilitar la vida de los agricultores ya que supone mayor atención a los pequeños detalles que un cultivo en suelo y más bien busca elevar la calidad de los productos para

que los agricultores puedan negociar un precio más alto. Una de las condiciones a tomar en cuenta es que las semillas deben germinar y llegar a la fase de plántula para luego ser plantadas en el sistema hidropónico, es en esta fase en la que se debe identificar si la planta es en verdad la que se desea cultivar y cosechar, una actividad que puede complicarse en especial para personas que no son expertas en el tema.

Si este proceso no se hace adecuadamente se corre el riesgo de confundir plantas similares como el maíz y el sorgo, los diferentes tipos de tomates existentes, diferentes tipos de papaya, entre otras, provocando un uso incorrecto de recursos, tiempo y espacio, además de la pérdida económica que generaría cultivar y tratar de vender un producto no deseado.

Alcance

Este proyecto tiene como fin presentar un algoritmo capaz de identificar entre al menos cuatro tipos de plántulas, de modo que, se iniciará buscando el método más atractivo para el análisis computacional de las imágenes. El aprendizaje de máquina se realizará a través del método de clasificación y para su entrenamiento se utilizará una base de datos de fotografías disponible en la internet, de la cual también se apartará una pequeña sección para la validación.

Justificación

El propósito de este proyecto es recopilar la información necesaria con la intención de crear un método de clasificación de plántulas empleando visión artificial y aprendizaje de máquina.

Es beneficioso ya que podría ser aplicado en el momento previo a la colocación de la plántula en un cultivo hidropónico. El método aseguraría a la persona responsable que efectivamente se trata de la planta requerida y así evitaría un consumo innecesario de recursos (tiempo y espacio), factores muy demandantes dentro de este tipo de cultivo.

Además, es un método que podría ser útil en otras aplicaciones, una de ellas es la identificación de malezas en las plantaciones convencionales, una problemática muy común que afecta la calidad y retrasa la producción del cultivo.

Capítulo 1

Revisión literaria

El uso de la visión artificial para la automatización de los procesos en la agricultura ha representado un cambio relevante en la mejora de calidad del producto final, lo que ha incentivado un aumento en el uso de este tipo de tecnologías durante todo el desarrollo de diferentes clases de plantas. Por lo que es importante empezar a conocer a fondo todo lo que compone los nuevos avances en la agricultura para poder ser partícipe de ellos.

1.1. Métodos de clasificación

1.1.1. Clasificación de Plántulas usando Aprendizaje Profundo

En este trabajo, se describe el programa diseñado para la clasificación de plántulas, en base a un conjunto de datos de aproximadamente 5,000 imágenes con 960 plantas pertenecientes a 12 especies en diferentes etapas de desarrollo. El algoritmo hace uso de una Red Neuronal Convolucional (CNN), una técnica de aprendizaje profundo ampliamente aplicada en el reconocimiento de imágenes, pues al usarse en la automatización agrícola se obtiene una clara mejora en la cosecha, producción y la productividad, siempre y cuando se diseñen adecuadamente. El modelo entrenado logró una precisión del 99,48 % en un conjunto de prueba extendido, lo que demuestra la viabilidad de este enfoque [1].

1.1.2. Monitoreo del Crecimiento de Plantas usando la Técnica de Contornos Activos

En este trabajo se analizan los efectos de algunos fertilizantes en las plantas usando Contornos Activos, una técnica de visión por computadora que calcula el crecimiento de las plantas tras detectar el contorno del objeto mediante el método de las serpientes haciendo uso de características físicas como elasticidad y flexión en la forma del objeto. Las semillas seleccionadas fueron frijol, calabacín, caupí y pepino debido a su alto nivel de producción. Su plantación se manejó inmediatamente después de la germinación, en macetas con la mezcla de tierra y algunos fertilizantes naturales y sintéticos como: nitrógeno, fosfato, potasio y compuestos. Después de las dos semanas de plantación se miden los efectos de los fertilizantes en el crecimiento capturando imágenes durante la fase de plántula que se analizan mediante el uso de GUI diseñada en C#. Esta investigación determinó que el tipo de fertilizante 18-46-0 conduce al mejor crecimiento de las plantas, en especial, pepino, frijol y calabacín, mientras que con los fertilizantes 15-15-15 y 46-0-0

no se consiguen valores satisfactorios que valgan la pena usarlos en los 4 tipos de plantas probadas [2].

1.1.3. Técnicas de Visión Artificial para la Segmentación y Detección de Líneas de Cultivo en Imágenes Agrícolas

Montalvo utiliza las técnicas de visión artificial para la clasificación de malas hierbas determinando la localización de las líneas de cultivo de maíz y la posición exacta de los rodales de mala hierba para proceder a la eliminación de éstos de forma precisa. En este trabajo se hace uso de dos Sistemas Expertos Automáticos (SEA): el primero diseñado para trabajar en campos agrícolas, aunque éstos posean una alta densidad de malas hierbas y el segundo, se plantea para dar solución a situaciones complejas en las que las plantas se encuentran enmascaradas de películas de barro debido a la lluvia o a la acción de regado artificial. Los sistemas propuestos se implementan en las tres unidades de robots que conforman la flota de vehículos RHEA (Robot Fleets for Highly Effective Agriculture and Forestry Management), que a su vez se protegen dentro de la cabina del tractor. Finalmente, se verificó su validez y eficacia para aplicaciones de tiempo real con los vehículos navegando a 3km/h a una altura no mayor a 5m, concluyéndose que el valor medio del tiempo requerido para procesar una imagen se sitúa en torno a los 400 milisegundos, siendo la ejecución en el peor de los casos de 570 milisegundos, dependiendo principalmente de dos factores, que son el estado de crecimiento del cultivo y el nivel de infestación de las malas hierbas existente. Este resultado permite que el vehículo realice labores de tratamiento mientras el vehículo navega [3].

1.1.4. Sistema de Inspección y Clasificación de Hojas de Plantas Medicinales por medio de Visión Artificial

En este trabajo se expone el desarrollo de un sistema para la clasificación de hojas de plantas medicinales (hinojo y caléndula), basado en visión artificial, con el fin de ser utilizado en una planta de extracción de aceites y en la elaboración de tisana. El algoritmo comienza tomando una fotografía y analizándola para identificar de manera automática la planta que se encuentra dentro del sistema, aunque también incluye la opción de operación manual, en la que el operario podrá ver la fotografía y seleccionar el tipo de planta que se encuentra dentro del sistema. Con ayuda de la misma fotografía se detecta el estado de la planta para decidir entre los 3 tipos de procesos el estado actual de la planta se categoriza en tres procesos (proceso 1: extracción de aceite, proceso 2: tisana, proceso 3: desecho), en esta etapa se aplica diferentes métodos de clasificación, tales como los algoritmos de segmentación de imágenes por medio de kmeans, Redes Neuronales MLP y Redes Neuronales Convolucionales, estos métodos están programados en Python y montados en un sistema embebido (Raspberry pi 3), por su facilidad de uso y su bajo costo. En todas las evaluaciones se obtuvo una precisión mayor al 50 %, factor que hizo factible el uso del algoritmo [4].

1.1.5. Sistema de Visión Artificial para el Análisis de Imágenes de Cultivo basado en Texturas Orientadas

En este trabajo se muestra el diseño de un sistema capaz de detectar la dirección de las líneas de cultivos de trigo de imágenes aéreas tomadas con ayuda de vehículos aéreos no

tripulados a diferentes alturas (30 y 100 metros), las cuales muestran ciertas características a las que el sistema se debe afrontar basándose en el análisis de texturas orientadas con 3 estrategias diferentes para el cálculo del ángulo global de orientación, logrando resultados en tiempo real consiguiendo procesar 15 tomas por segundo, manteniendo un alto nivel de eficiencia en la detección [5].

1.1.6. Sistema Automatizado de Reconocimiento de Malas Hierbas en Cultivos de Algodón basado en Aprendizaje

En este proyecto se describe la construcción de un sistema capaz de seguir al trabajador encargado del cultivo y advertirle mediante alarmas cuando se ha reconocido cenizo blanco o cenizo tornasol, malas hierbas típicas en el cultivo de algodón. El hardware de este proyecto se compone de un robot creado a partir del chasis de un coche capaz de dotar al sistema la capacidad de trasladarse a lo largo de los surcos de algodón. El movimiento se genera a partir de 4 motores y controladores para nivelar las intensidades que se deseen. Como fuente de energía se usa una batería de 12 voltios. Todo esto es controlado por Raspberry Pi que se alimenta de una batería portátil para móviles. Para el reconocimiento de las malas hierbas y para el seguimiento del trabajador se utilizan dos cámaras capaces de tomar 30 fotogramas por segundo. Para la elaboración del algoritmo necesario para la detección del trabajador, las malezas y la generación de alarmas, se hizo uso de la librería OpenCV, el módulo Object Detection. Finalmente, el algoritmo muestra una precisión de identificación del 66.66 % para el cenizo tornasol y del 100 % para el cenizo blanco siendo de 10ms el tiempo de procesamiento para la clasificación de malezas y 20ms para la identificación del trabajador [6].

1.1.7. Uso de la Visión por Computadora y Técnicas de Imágenes Multiespectrales para la Clasificación de Semillas de Caupí

El objetivo de este estudio fue investigar el potencial de la visión por computadora y los sistemas multiespectrales compatibles con el análisis multivariado para la clasificación de alto rendimiento de las semillas de caupí. Con este fin se utilizó un sistema automatizado de germinación por visión por computadora para el monitoreo ininterrumpido de semillas durante imbibición y germinación para identificar diferentes categorías de todas las semillas individuales. Mediante el uso de firmas espectrales de semillas de caupí extraídas de imágenes multiespectrales, diferentes modelos de análisis multivariados basados en la discriminación lineal. Se desarrollaron análisis de minant (LDA) para clasificar las semillas en diferentes categorías según el envejecimiento, la viabilidad, condición de plántulas y velocidad de germinación. Los resultados revelaron que los modelos LDA tenían buena precisión para distinguir semillas 'envejecidas' y 'no envejecidas' con una clasificación general correcta (OCC) de 97.51 %, 96.76 % y 97 %, semillas 'Germinadas' y 'No germinadas' con OCC de 81.80 %, 79.05 % y 81.0 %, semillas de 'Germinación Temprana', 'Germinación Media' y 'Muertas' con OCC de 77.21 %, 74.93 % y 68.00 % y entre semillas que dan plántulas 'Normal' y 'Anormal' con OCC de 68.08 %, 64.34 % y 62.00 % en entrenamiento [7].

1.1.8. Método de Clasificación de Plántulas basado en Transferencia de Aprendizaje

En este documento se propone un método de clasificación para plántulas basado en transferencia de aprendizaje. Se empezó extrayendo y atenuando la región de interés de la imagen original adquirida, una escala regional de grises acumulativa se obtiene la curva de distribución. Luego se calculó el número de puntos de pico de la curva para identificar la especificación de la bandeja de semillas. En segundo lugar, se utilizó el método de transferencia de aprendizaje basado en la red neuronal convolucional para construir el modelo de clasificación de plántulas. De acuerdo con las características de crecimiento de las plántulas, se recolectaron 2286 muestras de plántulas para entrenar el modelo. Finalmente, la imagen de la región de interés se divide en imágenes celulares de acuerdo con la especificación de la bandeja de semillas, y las imágenes de celda se colocan en el modelo clasificando así las plántulas calificadas, las plántulas no calificadas y la falta de plántulas. El método VGG16 es el que presentó una mayor precisión siendo del 95.50 % para tres especificaciones (50 celdas, 72 celdas, 105 celdas) de las plántulas de pimiento de 20 días y 25 días [8].

1.2. Resumen de los métodos de clasificación

En el cuadro 1.1 se puede destacar las características principales de los trabajos previos analizados, las cuales permiten tener una visión más general de los temas.

Trabajos	Métodos	Objetivo	Ambiente controlado	Base de datos propia	Plántas utilizadas	Hardware*	Resultado
1.1.1	CNN	Clasificación de plántulas	Si	No	* Bolsa de pastor * Mostaza de campo * Hierba negra * Manzanilla sin olor * Pamplina * Amor de hortelano * Trigo harinero * Remolacha azucarera * Pasto de viento * Maíz * Geranio silvestre * Cenizo	No	Precisión de identificación: 99,48%
1.1.2	Contornos activos	Determinar los efectos de ciertos fertilizantes	Si	Si	* Frijol * Calabacín * Caupí * Pepino	No	Fertilizante 18-46-0 con mayor desarrollo en el crecimiento de la plánta
1.1.3	* SEA * Método de Otsu	Control de malezas	No	Si	Cultivo de maíz	Si	Tiempo de procesamiento de imágenes: 400 - 570 ms
1.1.4	* MLP *CNN	Clásificación de hojas	Si	Si	* Hinojo * Caléndula	Si	Precisión de identificación: 50%

1.1.5	Texturas orientadas	Control de malezas	No	Si	Cultivo de trigo	Si	Procesamiento de imágenes: 15 tomas por segundo
1.1.6	OpenCV (Object Detection)	Control de malezas	Si	Si	* Cenizo blanco * Cenizo tornasol	Si	* Cenizo blanco: precisión de identificación del 100% * Cenizo tornasol: precisión de identificación del 66,66%
1.1.7	* Imágenes multiespectrales * LDA	Clasificación de alto rendimiento	Si	Si	Caupí	No	Precisión de identificación: * Germinadas: 96,76% * No germinadas: 81% * Germinación temprana, me dia y muertas: 68% * Normal y anormal: 62%
1.1.8	VGG16	Clasificación de alto rendimiento	Si	Si	Pimiento	Si	Precisión de identificación: 95,50%

*** Hardware fuera del sistema de visión artificial**

Cuadro 1.1: Cuadro comparativo de los trabajos previos relacionados al presente proyecto [1]-[2]-[3]-[4]-[5]-[6]-[7]-[8].

1.3. Propuesta

Como se observa en las secciones anteriores las investigaciones utilizan visión artificial para la clasificación de diferentes tipos de plantas. Gracias a esta información se sustenta resolver el problema de clasificación de cuatro plántulas que son: pamplina, maíz, bolsa de pastor y remolacha azucarera, ya que corresponden a especies comunes en el Ecuador. Para lo cual se hace uso de las ventajas de una red neuronal convolucional para la clasificación de las plántulas, pero también se complementa con un detector de objetos para crear un sistema de visión completo que ayude a los agricultores a optimizar su cosecha.

Capítulo 2

Metodología

Una de las líneas de investigación y desarrollo más interesantes es lo relativo a la "imitación" de los sentidos. Es por esto que la visión por computadora es una disciplina en creciente auge con multitud de aplicaciones [9] que dota al computador de la habilidad de percibir información para finalmente procesarla e interpretar la escena, siguiendo este comportamiento se han desarrollado una serie de pasos que los autores han generalizado en un marco de procedimientos que se llevan a cabo para reproducir tal acción definiendo 4 fases [10], las cuales se muestran en la Figura 2.1.

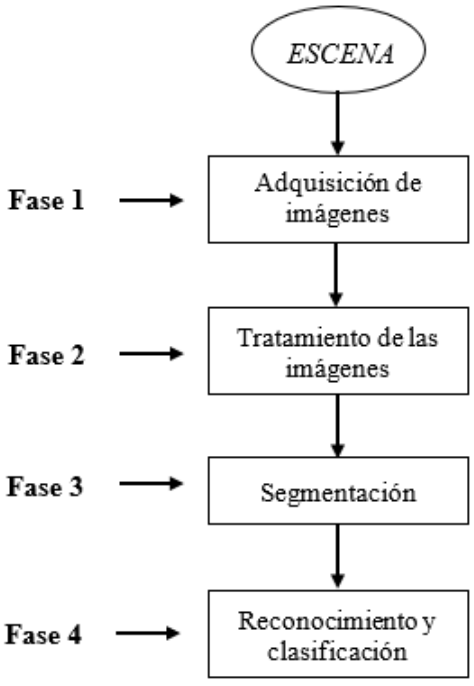


Figura 2.1: Diagrama de flujo de las fases de la visión artificial [9].

Debido a las necesidades y recursos disponibles para el presente trabajo el proceso de la Figura 2.1 se modifica, resultando en el diagrama de flujo de la Figura 2.2. Esta última muestra que el cambio inicia desde la adquisición de imágenes, reemplazando la etapa donde se establece un espacio para la creación de una base de datos propia por simplemente descargar una base de datos disponible en la internet[1]. Esta información es la

[1] Véase la sección 2.1.

base para la creación de una Red Neuronal Convolucional (CNN) capaz de clasificar entre dos plántulas de alimentos: maíz y remolacha dulce; y dos plántulas de malas hierbas: pamplina y bolsa de pastor[2]. Finalmente se aplica la detección de objetos basado en el clasificador de imágenes entrenado en la red convolucional.

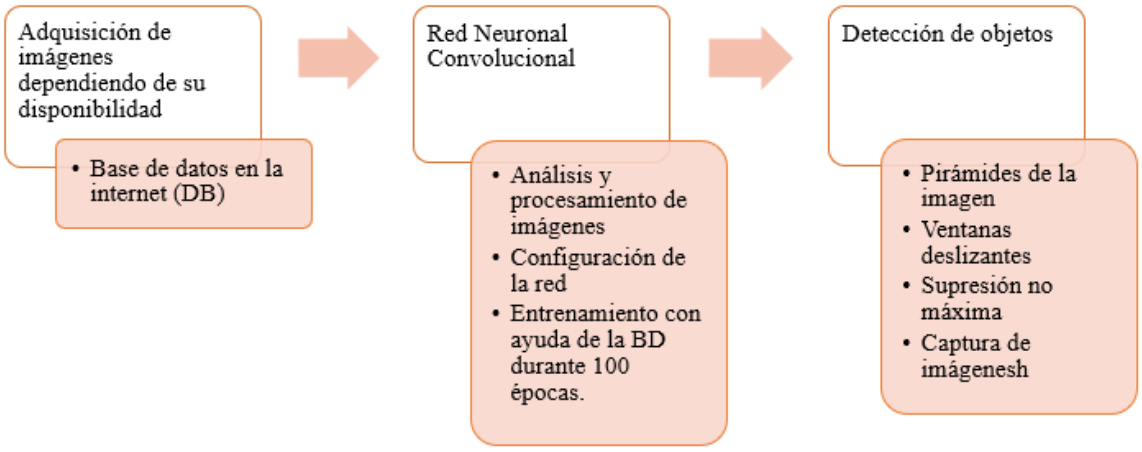
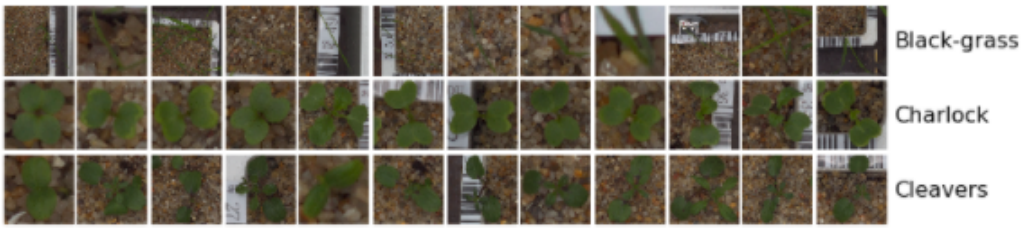


Figura 2.2: Diagrama de flujo del proceso aplicado para la creación del algoritmo del presente proyecto.

A continuación, a través de la sección 2.1 a la sección 2.3 se explica de manera detallada la aplicación de este algoritmo.

2.1. Adquisición de imágenes

La búsqueda se realiza en la plataforma Kaggle[3], en la cual se encuentran diversas opciones, sin embargo, cada opción presenta el mismo grupo de imágenes por lo que la selección de la base de datos se hace de manera aleatoria. La base de datos descargada [12] se compone de 5 544 imágenes RGB con una resolución física de aproximadamente 10 píxeles por mm y de formato .png, que comparten en su mayoría un mismo escenario de captura. Éstas corresponden a 12 clases de diferentes plántulas como se visualiza en la Figura 2.3. Las imágenes están divididas en dos carpetas llamadas “train” y “test” , que serán utilizadas más adelante, la primera para el entrenamiento[4] y la segunda para la evaluación[5] de la CNN.



[2]Véase la sección 2.2.

[3]Kaggle: plataforma web con un gran número de recursos en el campo de Machine Learning, disponibles de manera gratuita [11].

[4]Véase la sección 2.2.3.

[5]Véase la sección 3.1.

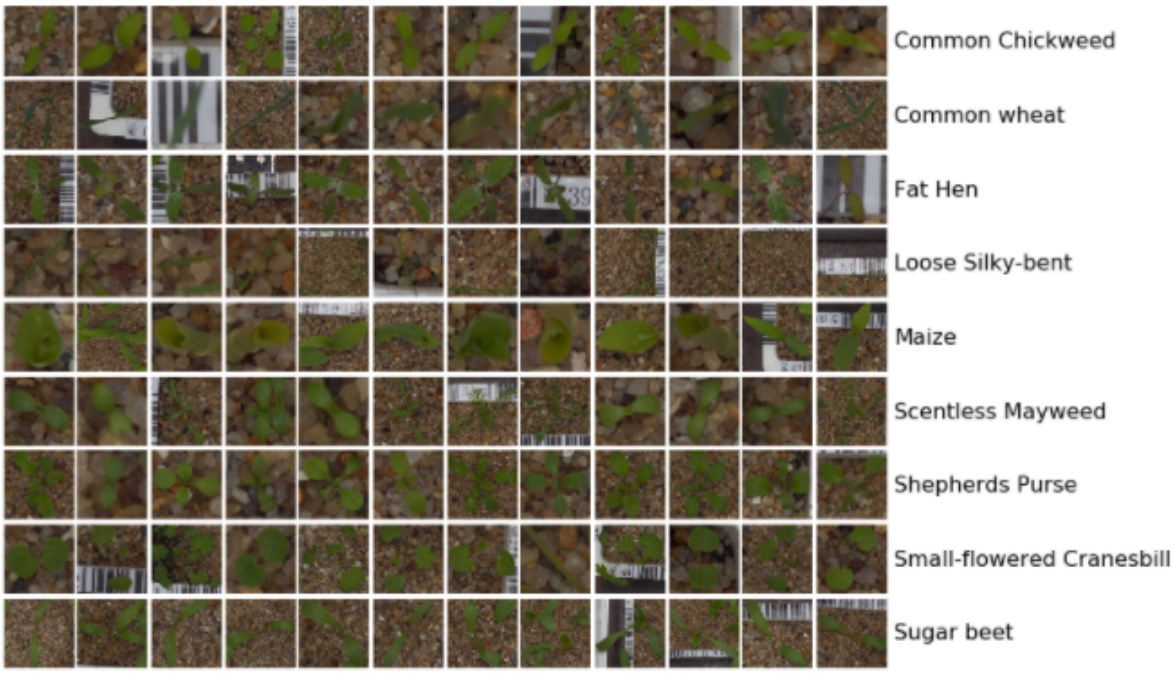


Figura 2.3: Ejemplos de cada clase contenidas en la base de datos original [13].

Con todos estos datos es necesario clasificar aquellos que cumplan con las necesidades del presente proyecto por lo que se analiza las características de cada clase.

Name	Nombre	Nombre Científico	Función	Cultivo afectado	Identificación	Apariencia	Presencia en Ecuador
Black Grass	Pasto negro	Alopecurus myosuroides	Maleza	Trigo	Flor	Similares a planta de cerales con flor en tonos verde oscuro, marrón y morado.	No
Charlock	Planta de campos	Sinapis arvensis	Maleza	Sinapis alba	Flor	Flor amarilla de cuatro pétalos que rodean semillas de color negro.	No
Cleavers	Amor de hortelano	Galium aparine	Maleza	Trigo, Sherardia arvensis	Hoja	Hojas agrupadas con pequeños ganchos.	No
Chickweed	Pamplina	Stellaria media	Maleza	Clavel, cacao	Flor	Flor blanca en forma de estrella.	Si
Common wheat	Trigo harinero	Triticum aestivum	Alimentación	-	-	Característico por tener una espiga de 40 granos.	Si
Fat hen	Quinhuilla	Chenopodium album	Maleza	Maíz, trigo, alfalfa	Hoja	Hojas de forma muy variable desde rómbico-ovadas a linear-lanceoladas, 2 veces más largas que anchas, enteras o dentadas	No
Loose silky bent	Pasto de invierno	Apera spica-venti	Maleza	Cereales	Flor	Las espiguillas son de una sola flor, con aristas cortas de tonos morados.	No
Maize	Maíz	Zea mays	Alimentación	-	-	Se caracteriza por tener una mazroca rodeada por las semillas de la planta.	Si
Scentless mayweed	Manzanilla sin aroma	Tripleurospermum inodorum	Maleza	Manzanilla, margarita	Flor	Las cabezas de las flores son blancas con un disco amarillo, no aromáticas.	No
Shepheerd purse	Bolsa de pastor	Capsella bursa-pastoris	Maleza	Cereales, papa	Fruto	Frutos en forma de corazón.	Si
Small flowered cranesbill	Geranio pequeño	Geranium pusillum	Maleza	Cereales	Flor	Flores generalmente en pares y de 5 pétalos, violeta claro, con puntas poco profundas.	No
Sugar beet	Remolacha azucarera	Beta vulgaris subsp. vulgaris var. altissima	Alimentación	-	-	Tipo de remolacha de la cual se obtiene azucar industrialmente.	Si

Cuadro 2.1: Cuadro comparativo. Sección verde) Características analizadas y las doce plántulas contenidas en la base de datos. Sección naranja) Plántulas seleccionadas para el desarrollo del presente proyecto [14]-[15]-[16]-[17]-[18]-[19]-[20]-[21]-[22]-[23]-[24]-[25]-[26]-[27]-[28]-[29]-[30]-[31]-[32]-[33].

Las características que analizar se muestran en el Cuadro 2.1 donde se incluye una

última columna para determinar su presencia en el Ecuador, siendo un factor decisivo para la selección de las cuatro plántulas, ya que el algoritmo busca solventar problemáticas del entorno. Se descarta la opción del trigo harinero con el fin de crear variedad entre los alimentos y no trabajar solo con cereales. Finalmente, las plántulas seleccionadas son: pamplina, maíz, bolsa de pastor y remolacha azucarera que corresponden a las filas de color naranja en el cuadro anterior.

Una vez se cuente con la lista de plántulas a utilizar, se eliminan los datos del resto de plántulas de las carpetas "train" y "test" para crear una nueva base de datos que solo cuente con las cuatro plántulas del listado. Algo que es importante realizar en esta etapa, con el fin de crear un modelo óptimo en su nivel de predicción y evitar el fenómeno del sobreajuste, es crear una tercera carpeta denominada "validation", en la que se almacene el 20 % de los datos de la carpeta "train" que ahora se quedará con el 80 % restante (Figura 2.4).

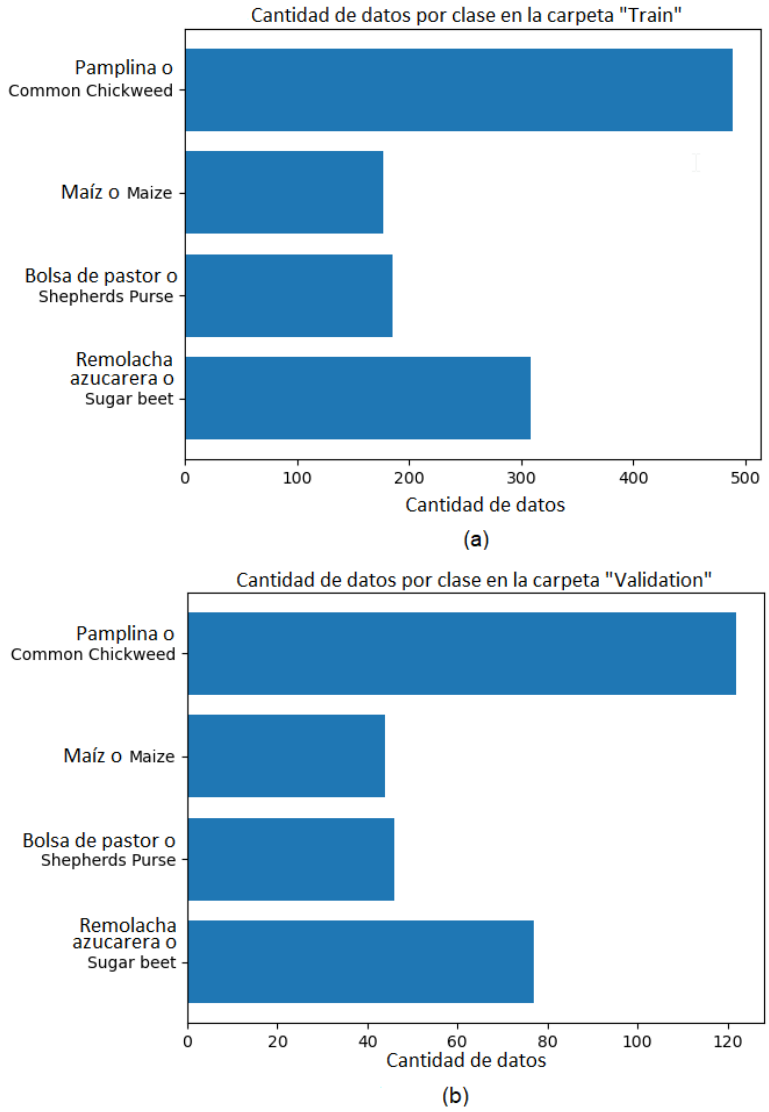


Figura 2.4: a) Gráfica de barras de la cantidad de imágenes contenidas en cada clase de la carpeta de "train". b) Gráfica de barras de la cantidad de imágenes contenidas en cada clase de la carpeta "validation".

Estas carpetas deben conservar la estructura de clasificación donde cada clase está en una carpeta como se ve en la Figura 2.5 de manera obligatoria ya que es una condición para la compilación del algoritmo debido a la función ImageDataGenerator[6]. En este punto se puede cambiar los nombres de las carpetas, sin embargo, para este proyecto se mantienen, por lo que las etiquetas en los objetos estarán en inglés, igual al nombre de la carpeta de cada clase.

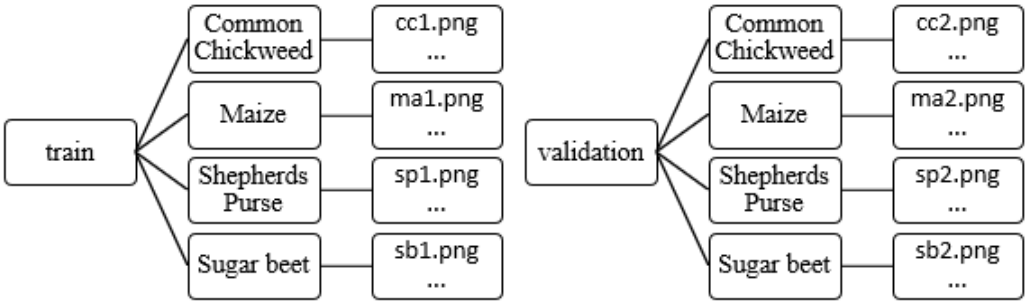


Figura 2.5: Estructura de las carpetas de entrenamiento y validación para el correcto funcionamiento del algoritmo.

Con respecto a la carpeta "test" es necesario modificar su estructura a la que se muestra en la Figura 2.6, ya que de igual a las carpetas train y validation, es una condición obligatoria de la sección 2.2.1.

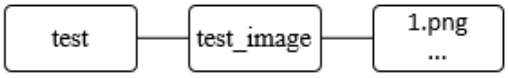


Figura 2.6: Estructura de la carpeta evaluación.

2.2. Creación de la Red Neuronal Convolucional (CNN)

Al realizar un estudio de las características y ventajas de utilizar redes neuronales, se confirma que son las más populares para la tarea de clasificación de imágenes, pues generan poderosos modelos visuales con jerarquías de características que permiten una segmentación precisa por lo que realizan predicciones relativamente más rápidas que otros algoritmos mientras mantienen un rendimiento competitivo [34], todo esto gracias a la estructura que se puede observar en la Figura 2.7. Además, en el trabajo previo [1] se muestra como tras al usar una CNN[7] se obtienen resultados favorables en la clasificación de doce plántulas.

[6] Véase la sección 2.2.1.

[7] Las CNN tienen las capacidades de extraer características generalmente útiles de los datos con o sin etiquetas, detectan y eliminan redundancias de entrada y preservan solo aspectos esenciales de los datos en representaciones sólidas y discriminativas; pueden capturar las características más obvias de los datos, por lo que podrían lograr mejores resultados en varias aplicaciones. A diferencia de las características creadas a mano, como SIFT y HOG; las características extraídas por la CNN se generan de extremo a extremo, lo que elimina la intervención humana. Todo esto gracias al menor número de conexiones y parámetros lo que favorece que la extracción de características sea más eficiente [34].

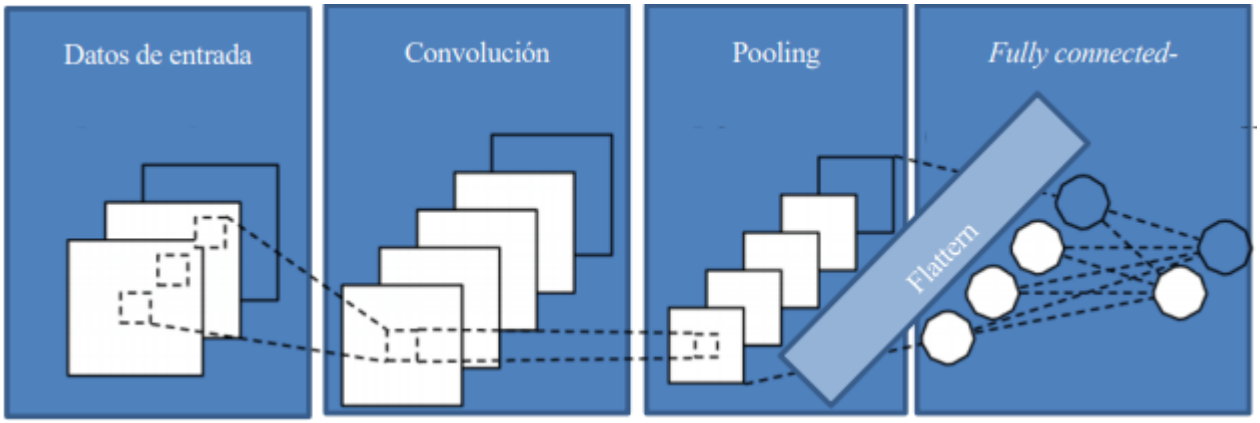


Figura 2.7: Estructura general de una Red Neuronal Convolucional [35].

Con base a esta información se decide trabajar con el código adjunto a la base de datos seleccionada [36]. Este algoritmo corresponde a una red neuronal convolucional capaz de clasificar entre doce tipos de plántulas, entre ellas, las cuatro seleccionadas para este proyecto, por lo que se toma como base para la creación de una nueva CNN.

2.2.1. Análisis y procesamiento de imágenes

Como ya se ha mencionado anteriormente, la base de datos se compone de imágenes RGB, el modelo más utilizado en la creación de colores en pantalla. Además de ser el formato que utiliza las cámaras. En este espacio de color las intensidades de la luz relativas al rojo, verde y azul son sumadas para la creación de colores que incluyen el negro y el blanco [37]. Es con este modelo que se inicia el análisis de las imágenes, aislando la zona relevante, es decir, la zona donde se ubica la plántula, para luego procesar de mejor manera la información resultante y aumentar la velocidad del entrenamiento de la CNN.

Análisis de color

Para iniciar con el análisis de color, el autor [36] destacó las dificultades del uso de los datos seleccionados ya que no todas las imágenes cuentan con una resolución constante y además no parecen ser tomadas en un mismo día, más bien son capturadas en varios puntos de crecimiento, sin que esto signifique que la información obtenida no corresponda a la fase de plántula. A pesar de esto, el factor relevante es el fondo de la mayoría de las imágenes (Figura 2.3) que al ser similar facilita su eliminación conservando solo la información necesaria para el entrenamiento. Para aprovechar esta ventaja se toma una muestra aleatoria de 50 píxeles de 10 imágenes de las 4 clases seleccionadas (Figura 2.8), la cual es utilizada para trazar una gráfica en el espacio de color RGB como se muestra en la Figura 2.9.

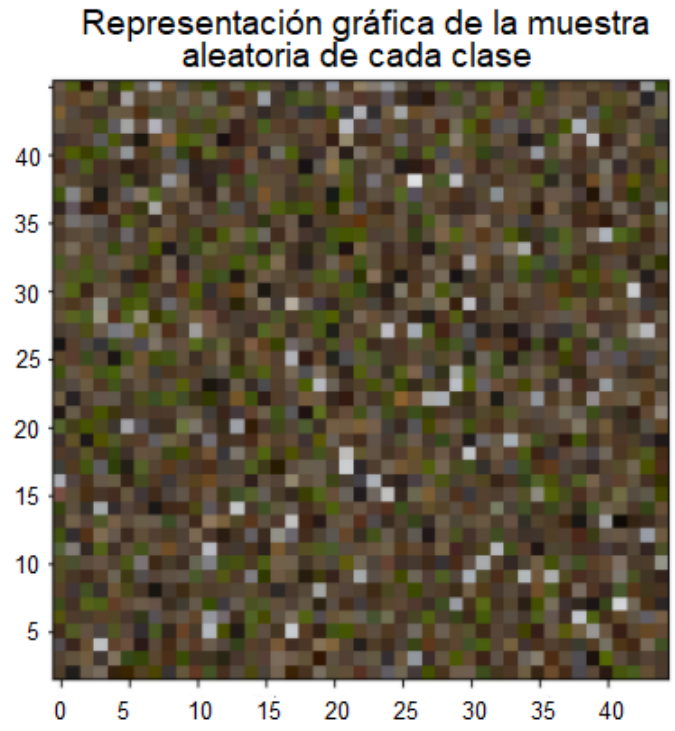


Figura 2.8: Representación gráfica de la muestra aleatoria de 50 píxeles de 10 imágenes de cada clase dentro de la base de datos modificada.

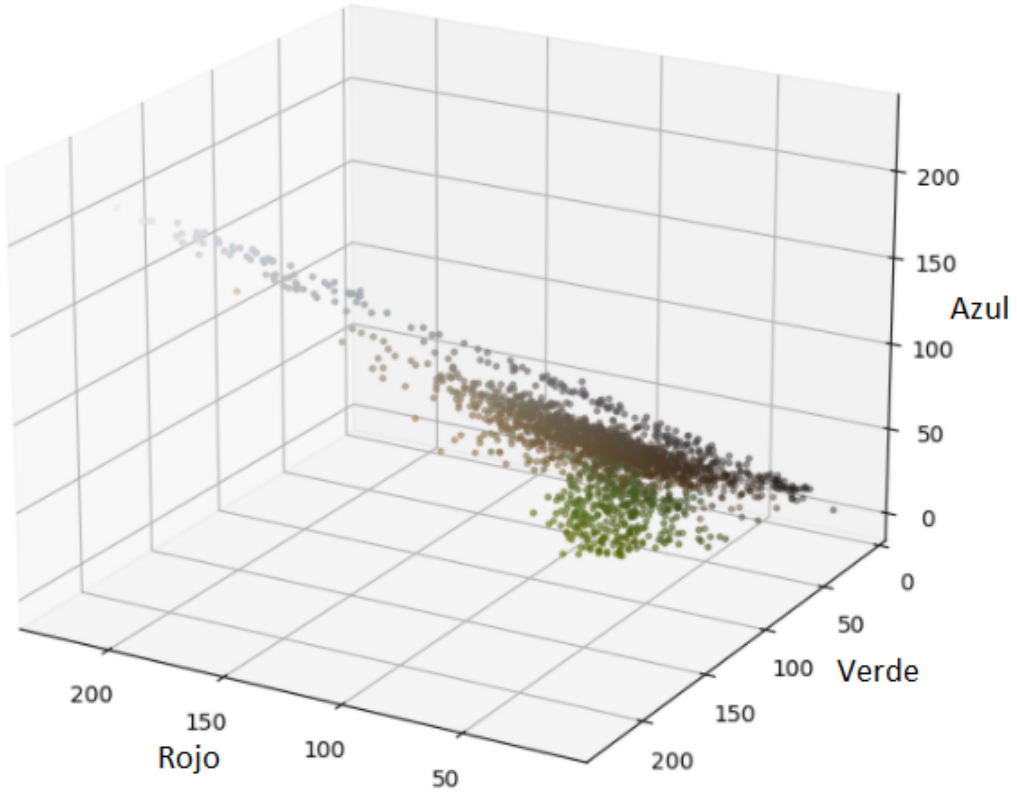


Figura 2.9: Representación de la muestra en píxeles dentro del espacio de color RGB.

Al analizar la Figura 2.9 se notan zonas que se componen de colores blanco, negro, gris y café que representan el fondo presente en todas las imágenes de manera muy similar, mientras que el resto se muestra en color verde que representa a las hojas de las plántulas, ambas zonas parecieran poder ser separadas, sin embargo, al seleccionarse los valores de los límites inferior y superior, no se obtiene un buen resultado pues al observar más de cerca la distribución de los píxeles es posible notar pequeñas zonas en las que no solo está el color verde sino que esta sombreado suavemente por otros colores, causando pérdida de información al momento de la separación, resultando en imágenes de plántulas sin fondo, pero con partes de sus hojas incompletas.

Antes de empezar a buscar complejos métodos para obtener la mayor información dentro del espacio de color RGB, se intenta con la misma técnica del autor [36], evaluar la misma muestra de 50 píxeles en un espacio de color de modelo HSV[8] ya que al contrario del modelo RGB, representa las tonalidades de colores mayormente agrupadas (Figura 2.10) [38].

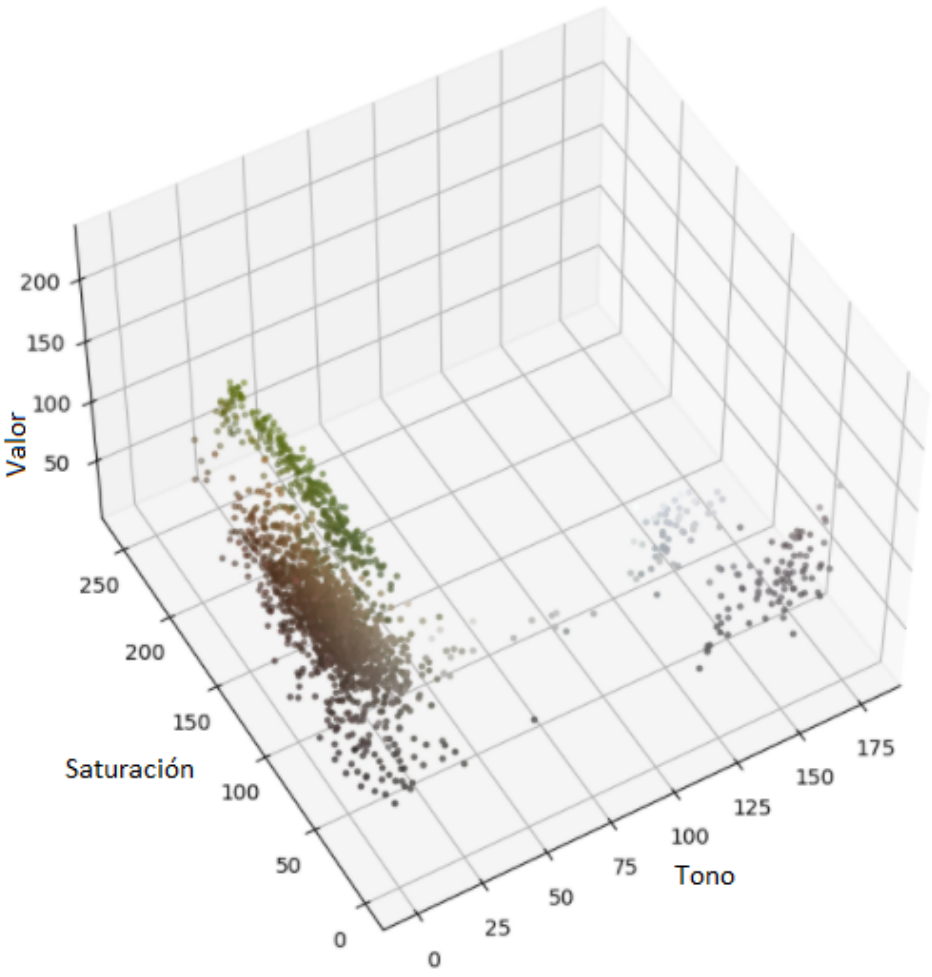


Figura 2.10: Representación de la muestra en píxeles dentro del espacio de color HSV.

[8]Modelo HSV: es un modelo derivado del modelo de color RGB. Con esta representación del estado del color, cada color trabaja con 3 componentes básicas: Tono(Hue) que hace referencia al valor de la cromaticidad o clase de color, Saturación (Saturation) se refiere a las longitudes de onda que se suman a la frecuencia de color, y determinan la cantidad de blanco que contiene un color y Valor (Value) que representa la apreciación subjetiva de claridad y oscuridad [38].

Gracias al cambio del modelo de color, en la gráfica de la Figura 2.10 se nota una distribución de los píxeles más conveniente para su separación, por ejemplo, se puede ver más claramente cuáles son los datos de color verde que representan a las plántulas y también los datos de colores marrones, blancos y grises que representan el suelo, piedras, etiquetas y otros elementos presentes en el fondo de todas las imágenes.

Detección de bordes

Ya establecido el modelo HSV para la extracción de características, se establecen los límites inferiores y superiores de las zonas verdes y la zona que contiene el resto de los colores. Para esto, se puede tomar dichos valores analizando la Figura 2.10, sin embargo, en ella se puede apreciar que los valores en el eje de Valor casi permanecen constates por lo que se opta por volver a graficar la información omitiendo este eje (Figura 2.11).

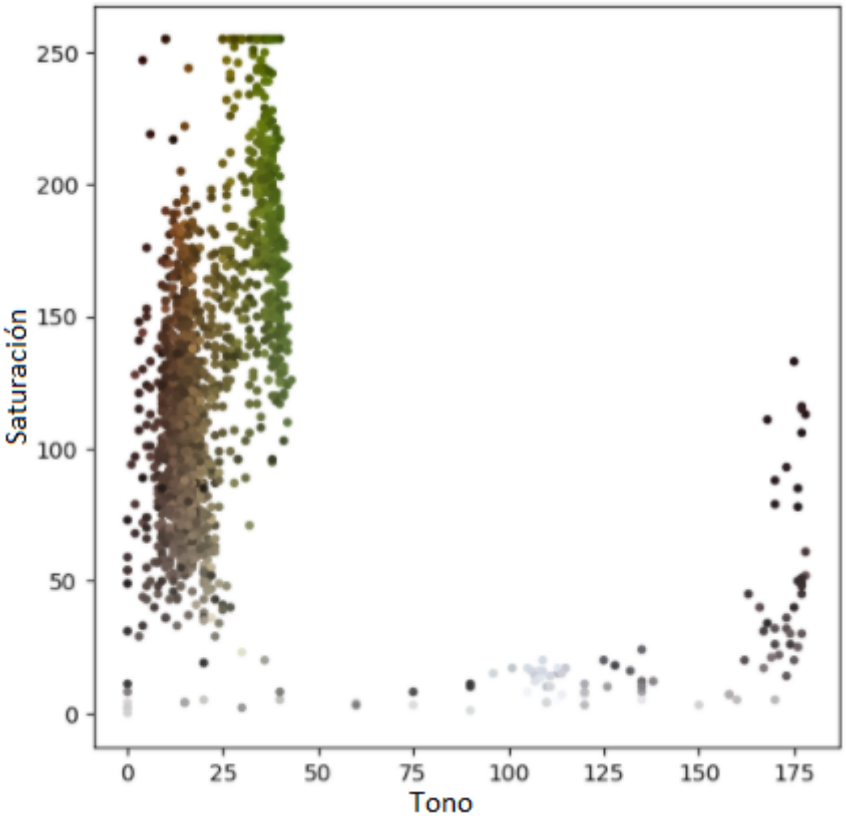


Figura 2.11: Representación de la muestra en píxeles dentro del espacio de color HS.

En la nueva representación de la muestra de la Figura 2.11 se seleccionan los límites superiores e inferiores con mayor precisión, los cuales aíslan los píxeles que estén fuera de los rangos de tono de 24 a 55 y de saturación de 50 a 255. Con estos datos y la librería de uso libre OpenCV[9] se pone a prueba todo el análisis del color realizado, tapando el fondo de una imagen aleatoria de cada clase.

[9] OpenCV: es una librería dirigida fundamentalmente a la visión por computador en tiempo real. Entre sus muchas áreas de aplicación destacarían: interacción hombre-máquina (HCI); segmentación y reconocimiento de objetos; reconocimiento de gestos; seguimiento del movimiento; estructura del movimiento (SFM); y robots móviles. [39].

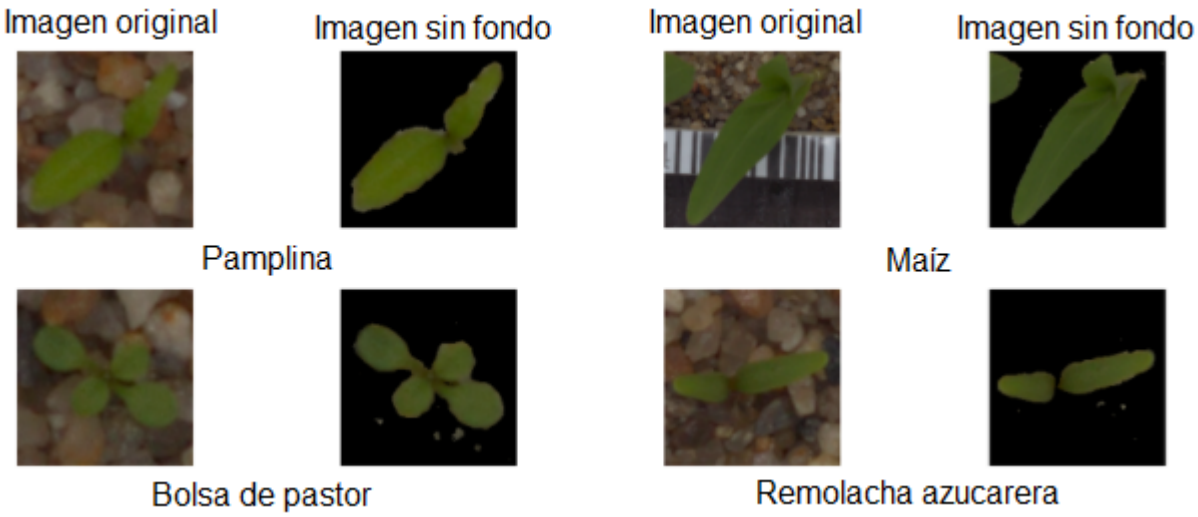


Figura 2.12: Comparación de las imágenes de cada clase con y sin fondo.

Como se muestra en la Figura 2.12 el resultado es óptimo por lo que se puede aplicar las mismas técnicas en el resto de las imágenes.

ImageDataGenerator

A pesar de que anteriormente se demostró que las técnicas implementadas brindan un resultado favorable, no podrá aplicarse ya que antes es necesario que la transformación del cambio de modelo de color sea compatible con el objeto ImageDataGenerator[10] de Keras[11] que será utilizado en la configuración de la CNN[12]. Para poner en marcha este objeto es necesario configurar la información de la base de datos en carpetas como se muestra en las Figuras 2.5 y 2.6.

Con la información lista se aprovecha el objeto ImageDataGenerator para realizar actividades adicionales a las establecidas, como son: reescalar los valores máximos de los píxeles a (0-1), aplicar giros y rotaciones aleatorias y añadiendo también la función de la eliminación del fondo de la Sección 2.2.1, en las imágenes de la carpeta train. Con respecto a las imágenes contenidas en las carpetas validation y test serán utilizadas por un generador diferente, el cual solo aplicará la función de eliminar el fondo de ellas y el mismo reescalado de los rangos máximos establecidos anteriormente. Al terminar el procesamiento de las imágenes se obtiene imágenes de tamaño 150x150, sin fondo y con los giros mencionados anteriormente, estas nuevas imágenes son la entrada que recibe la CNN para su posterior entrenamiento. Pero, este proceso no se completará hasta añadir al generador el valor del batch[13], dato que se obtendrá más adelante, antes de iniciar el entrenamiento de la CNN.

[10]ImageDataGenerator: es una función que genera lotes de datos de imágenes de tensores con aumento de datos en tiempo real. Los datos se repetirán (en lotes) de forma indefinida [40].

[11]Keras: es una biblioteca de Redes Neuronales de código abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow, Microsoft Cognitive Toolkit o Theano. Está especialmente diseñada para posibilitar la experimentación en más o menos poco tiempo con redes de Aprendizaje Profundo. Sus fuertes se centran en ser amigable para el usuario, modular y extensible [41].

[12]Véase la sección 2.2.2.

[13]Véase la ecuación 2.1.

Otra aplicación del resultado del generador aplicado a la carpeta train es el cálculo del valor de los pesos de las cuatro clases, los cuales son de gran importancia al entrenar la CNN[14] ya que el peso se aplica al valor de pérdida para la clase correspondiente. Los resultados del cálculo mantienen relación con la cantidad de datos de cada clase, entonces de igual manera que en la Figura 2.4, en los pesos también se puede notar un desequilibrio entre todas las clases.

Índice	Etiqueta	Pesos
0	Common Chickweed	0.50
1	Maize	1.38
2	Shepherds Purse	1.32
3	Sugar beet	0.79

Cuadro 2.2: Pesos calculados durante el entrenamiento de la CNN para cada una de las clases.

Como se muestra en el Cuadro 2.2, los pesos obtenidos son valores cercanos a 1, pero con variantes que pueden perjudicar el entrenamiento, pues un valor más alto coloca a la clase en un nivel de prioridad más alta, reduciendo la importancia del resto de clases, un inconveniente que será corregido en la sección 2.2.3. Es importante mencionar que con el cálculo de los pesos también se establecen las etiquetas que usa el detector de objetos para indicar el tipo de plántula a la cual corresponde, éstas corresponden a la columna "Pesos" del cuadro anterior.

2.2.2. Configuración de la CNN

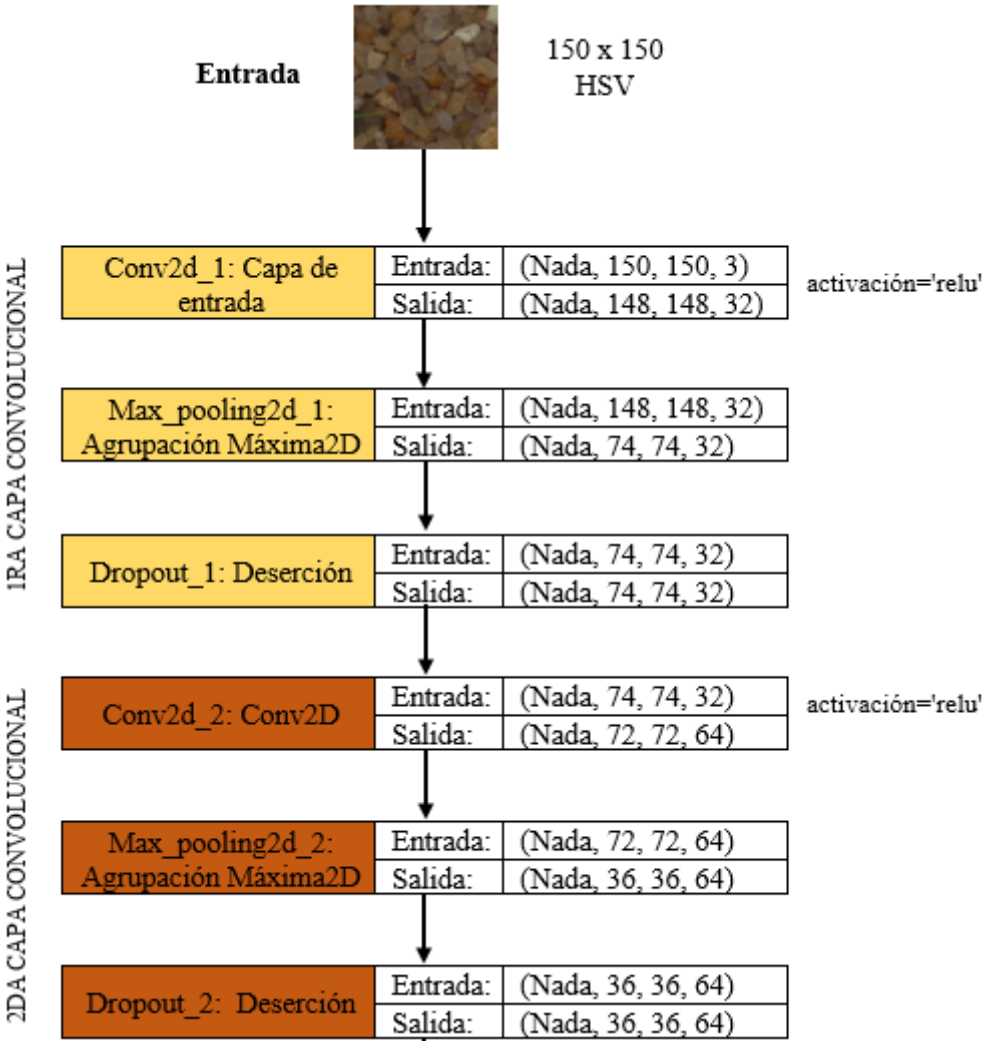
El modelo creado en [36] fue el resultado de varias modificaciones como:

- Número de capas
- Uso de normalización por lotes (batch)
- Horarios de tasas de aprendizaje
- Diferentes optimizadores (rmsprop, SGD, adadelta, etc.)
- Uso de dropout
- Nodos por capa
- Función de activación (relu con fugas)
- Tamaño de la imagen de entrenamiento (reajuste a 300x300 en lugar de 150x150)
- Inclusión / exclusión de la eliminación de antecedentes

Después de las cuales, solo el modelo con el mejor resultado durante el entrenamiento es almacenado y publicado, por lo que es el mismo utilizado en el presente proyecto. Este

[14] Véase la sección 2.2.3.

modelo está formado por cuatro capas convolucionales[15] seguidas de una capa densamente conectada[16]. Incluye la deserción (dropout)[17] en cada capa para evitar el sobreajuste[18] Las funciones de activación de Relu[19] se utilizan para todas las capas excepto la última que requiere una activación softmax[20] para producir un resultado apropiado para la clasificación multiclase. Toda esta definición se puede apreciar de mejor manera en la Figura 2.13.



[15]Capa convolucional: La operación de convolución es la que da nombre a la arquitectura y se emplea en al menos una de sus capas, tanto a la imagen completa como a los mapas de características intermedias, generando nuevos mapas de características [35].

[16]Capa densamente conectada: se encarga de aprender patrones globales en su espacio global de entrada. Se posiciona al final del modelo para alimentar la capa final de softmax [42].

[17]Dropout: capa que regulariza el sobreajuste creando una red más robusta a los datos de entrada imprevistos, y solo están activas durante la etapa de entrenamiento de la red, es decir, no están presentes durante la etapa de predicción [34].

[18]Sobreajuste: tendencia de la red convolucional a memorizar datos de entrenamiento, generando porcentajes bajos de generalización [34].

[19]Función de activación ReLU (Rectified Lineal Unit): es una función que anula los valores negativos y deja los positivos tal como entran [43].

[20]Función de activación Softmax: se basa en calcular "las evidencias" de que una determinada imagen pertenece a una clase en particular y luego estas evidencias se convierten en probabilidades de que pertenezca a cada una de las clases [42].

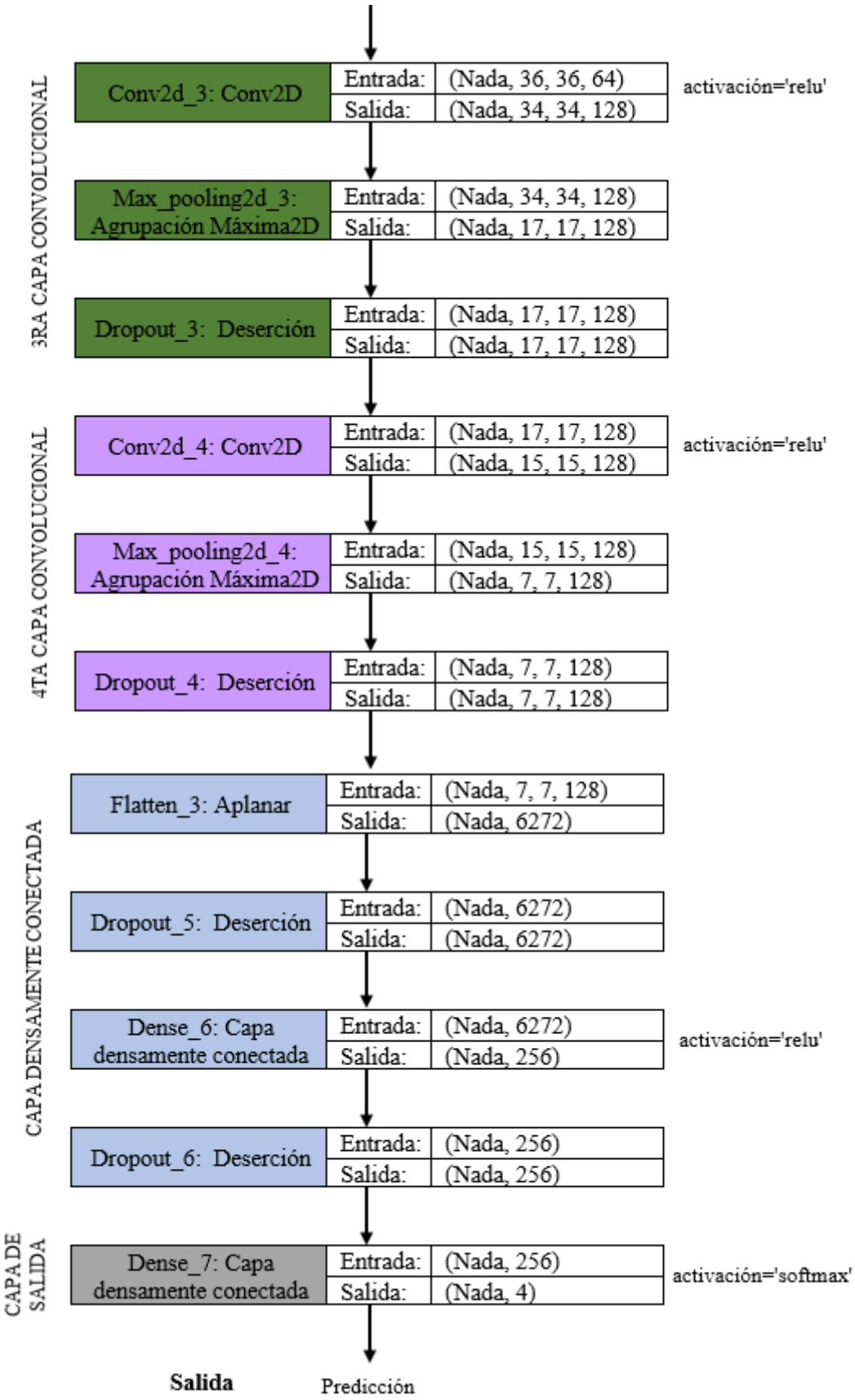


Figura 2.13: Esquema de la Red Neuronal Convolucional utilizada en el presente trabajo.

2.2.3. Entrenamiento

El modelo está listo para ser entrenado con las imágenes contenidas en las carpetas train y validation, sin embargo, antes es necesario corregir la falla que quedó pendiente en la Sección 2.2.1 para la cual se hace uso del método "fit generator" que es capaz de proporcionar ponderaciones de clase por medio de la frecuencia inversa normalizada, esto evitará que la CNN prefiera cierta clase solo por su número de registros.

Como se puede apreciar en [36] el método "fit generator" se compone de ciertas variables que para este proyecto son modificadas, pues dependen de la cantidad de datos a utilizarse durante el entrenamiento de la CNN. Éstas variables son: los generadores de las carpetas train y validation[21], el resultado del cálculo de los pesos (Figura 2.2), pasos por época (steps per epoch)[22], pasos de validación (validation steps)[23] y épocas (epochs)[24]. Éstos tres últimos se calculan de la siguiente manera:

- Pasos por época: para conocer este valor, se inicia estableciendo el valor del lote(batch)[25], que al consultarlo en la literatura se descubre que los valores más usados dentro de tutoriales para la creación de CNN son 32, 64 y 128 [44], sin embargo, estos valores no son fijos y tampoco representan una garantía de obtener un resultado positivo, una clara prueba es el algoritmo desarrollado en [36] que utiliza un batch=20.

  Para el presente trabajo se realiza el entrenamiento, con todos los valores típicos obteniendo resultados de precisión mayores al 90 %, sin embargo, en busca de aumentar los pasos por época se escoge un lote de menor valor, entonces el nuevo lote es igual a 16. Por lo que ahora se puede aplicar la ecuación 2.1 para calcular los pasos por época.

$$\text{pasos por época} = \frac{\text{cantidad de imágenes en la carpeta train}}{\text{lote}} \tag{2.1}$$

$$\text{pasos por época} = \frac{1159}{16} \tag{2.2}$$

$$\text{pasos por época} = 72,44 \approx 72 \tag{2.3}$$

  Como se aprecia en la ecuación 2.3 el valor que se asigna a los pasos por época no se aproxima ya que en la realidad para llegar al 73 debería crearse una mitad de imagen. Con el valor de 72 se obtiene un aumento en la precisión del modelo.

- Pasos de validación: una vez calculado los pasos por época, basta con repetir el proceso anterior, reemplazando la cantidad de datos y conservando el mismo lote, como se ve en la ecuación 2.4.

[21] Véase la sección 2.2.1

[22] Pasos por época: representa la cantidad de iteraciones por lotes dentro de cada época. Su uso es conveniente cuando se trabaja con un gran número de datos dentro de una carpeta [44].

[23] Pasos de validación: es similar a los pasos por época con la diferencia de trabajar con los datos contenidos en una carpeta diferente [44].

[24] Épocas: es un hiperparámetro que define el número de veces que el algoritmo de aprendizaje funcionará a través de todo el dataset de entrenamiento [44].

[25] Lote: es un hiperparámetros que define el número de muestras por las que se va a trabajar antes de actualizar los parámetros internos del modelo [44].

$$\text{Pasos de validación} = \frac{\text{cantidad de imágenes en la carpeta validation}}{\text{lote}} \tag{2.4}$$

$$\text{Pasos de validación} = \frac{289}{16} \tag{2.5}$$

$$\text{Pasos de validación} = 18,06 \approx 18 \tag{2.6}$$

- Época: de acuerdo a la investigación realizada, típicamente los valores utilizados sobrepasan los cientos o miles para darle al algoritmo la oportunidad de ejecutarse hasta que el error del modelo se haya minimizado lo suficiente [44]. Es así que en el entrenamiento la época toma el valor de 100.

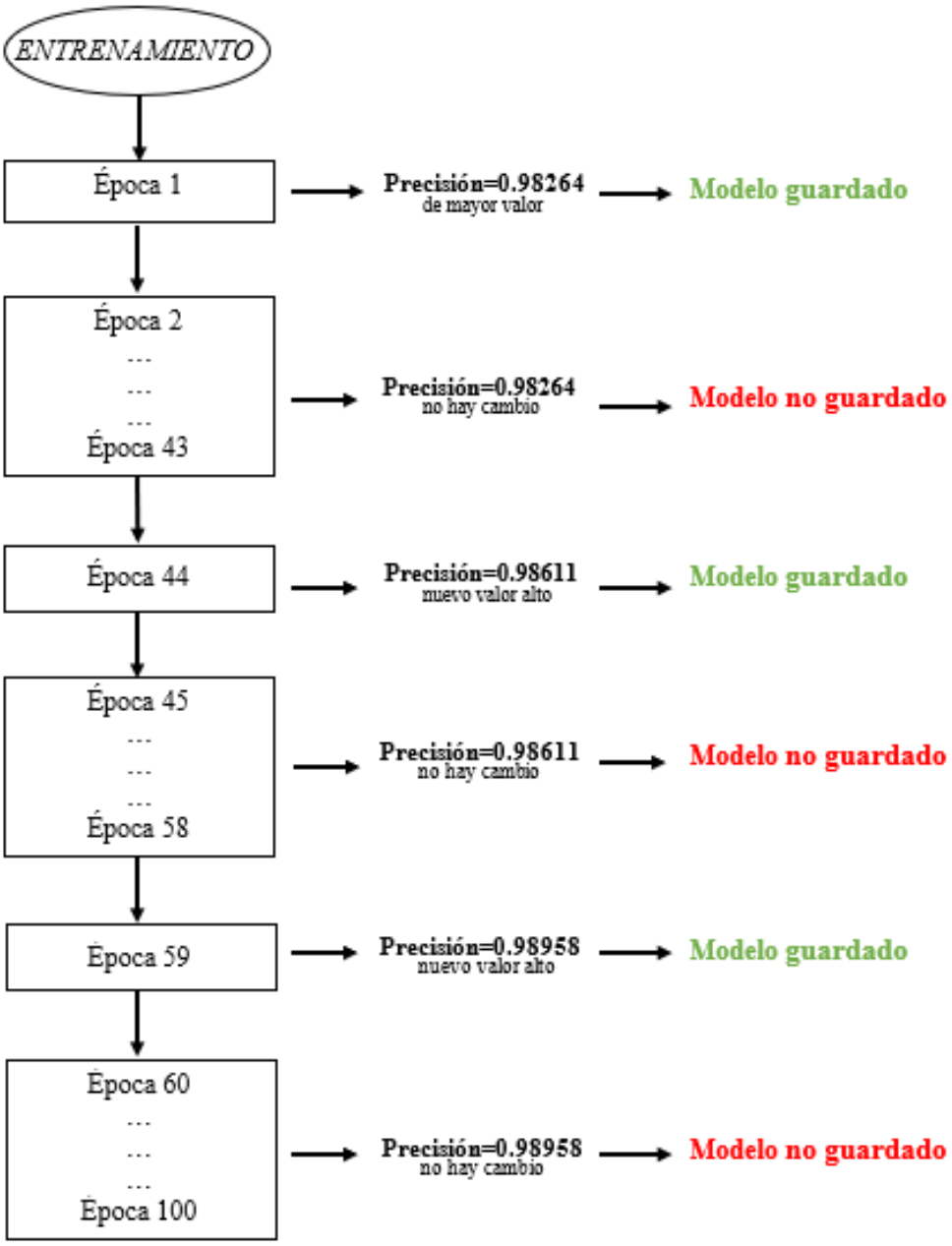


Figura 2.14: Resultados del entrenamiento de la CNN.

A pesar de establecer la época=100, en este proyecto la precisión de mayor valor se consigue en la época 59 como se muestra en la Figura 2.14, por lo que, siguiendo la recomendación de [36], una época de 60 hubiera funcionado de igual manera, lo que también disminuiría el tiempo de entrenamiento.

Con el fin de optimizar el entrenamiento se utiliza el objeto ModelCheckpoint que es capaz de ignorar modelos con resultados irrelevantes y solo almacenar aquellos en los que la precisión haya aumentado. También se añade un optimizador llamado Adam que se encarga de ajustar dinámicamente la tasa de aprendizaje en el entrenamiento [36].

Finalmente, al cumplir con todas estas condiciones, se pone en marcha el entrenamiento y como se ve en la Figura 2.14 las 100 épocas se componen de 72 pasos, lo que significa que el modelo trabaja con grupos de 72 imágenes por 100 veces, teniendo un total de 7 200 lotes durante todo el proceso [44]. Con toda esta cantidad de datos procesados es natural que el entrenamiento tome 3 horas.

2.3. Detección de objetos

Al terminar el entrenamiento de la CNN se obtiene un modelo capaz de clasificar las imágenes de tipo .png, sin embargo, es momento de utilizar este modelo para crear el sistema de visión artificial, es así como se opta por crear un algoritmo para la detección de objetos que, a diferencia de la clasificación de imágenes, además de asignar la etiqueta a una imagen también se asigna un cuadro delimitador que indica el lugar del objeto.

Para el diseño de este algoritmo se trabaja especialmente con el modelo clasificador de imágenes y de funciones básicas de los detectores como pirámides de la imagen[26], ventanas deslizantes[27] y supresión no máxima[28].

2.3.1. Pirámides de la imagen

Al ingresar la imagen en el programa, esta empieza a ser procesada por la función "image pyramid" que es configurada con variables como el valor de la escala con la que las imágenes irán reduciendo su tamaño y el "minSize" que representa el tamaño mínimo de la imagen a la que la función debe llegar para evitar que el ciclo sea infinito. La operación que se realiza para reescalar las imágenes se muestran en la ecuación 2.7.

$$\text{nuevo tamaño de la imagen} = \frac{\text{ancho de la imagen}}{\text{escala}} \tag{2.7}$$

El valor resultante representa el nuevo tamaño de la imagen que es de menor tamaño, este procedimiento se irá repitiendo hasta que llegue al tamaño del "minSize".

[26]Pirámides de la imagen: es una función que permite encontrar objetos en imágenes a diferentes tamaños. En la parte inferior de la pirámide se ubica la imagen original en su tamaño original y en cada capa subsiguiente, la imagen cambia de tamaño hasta que se cumple algún criterio de detención [45].

[27]Ventanas deslizantes: es un rectángulo de tamaño fijo que se desliza de izquierda a derecha y de arriba a abajo dentro de una imagen con el fin de ubicar el objeto a detectar [45].

[28]Supresión no máxima (NMS): al utilizar las ventanas deslizantes varios recuadros aparecen para rodear varias zonas de la imagen por lo que la NMS elimina los recuadros con menor probabilidad de detectar algo y también aquellos que se encuentran sobrepuestos, con el fin de mantener aquellos recuadros con mayor probabilidad de detectar un objeto [45].

2.3.2. Ventanas deslizantes

A la vez que se va creando la pirámide también se pone en acción la creación de ventanas deslizantes en cada una de las imágenes reescaladas, para esta acción se necesita configurar los siguientes parámetros:

- WIN STEP: este valor equivale al paso con el que el ROI SIZE se desplaza, tanto en el eje x como en el eje y. Tomando en cuenta que en este caso solo se analiza un pequeño grupo de imágenes no existe problema si el WIN STEP es pequeño, pero si no fuera el caso, este debe analizarse ya que puede provocar que se creen un número excesivo de ventanas. Tradicionalmente el valor de este paso corresponde a 4 u 8 píxeles [45], en este proyecto se aplica un paso de 16 mostrando resultados óptimos.
- ROI SIZE: indica el tamaño del recuadro que cubre un grupo de píxeles y que se desliza de izquierda a derecha, como se ve en la Figura 2.15. En este proyecto el tamaño es igual al del "minSize".

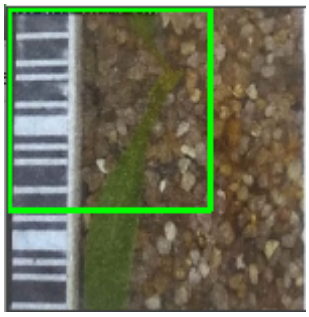


Figura 2.15: Ventana deslizante en una imagen de la base de datos modificada.

En cada zona donde se ubica el recuadro el algoritmo extrae la región de interés (ROI) para enviarla al modelo entrenado previamente. Se finaliza las funciones de pirámides de la imagen y ventanas deslizantes y después de ciertos segundos la CNN entrega resultados para todas las ROI, los cuales son decodificados para establecer un orden según su nivel de importancia.

2.3.3. Supresión no máxima

En esta sección se toman las predicciones decodificadas para eliminar aquellas de menor rango y las sobrepuestas. Posteriormente la información se muestra en una nueva ventana emergente (Figura 2.16), estos datos son: la imagen de entrada y la imagen de salida con el recuadro que marca el objeto y una etiqueta que lo describe.

Figura 2.16: Resultados del algoritmo detector de objetos. a) Ventana con la imagen de entrada y una ventana deslizante. b) Ventana con la imagen de salida, una ventana deslizante y la etiqueta correspondiente al objeto. c) Ventada de Anaconda Prompt en la cual se ejecuta el código y se visualizan resultados.

2.3.4. Captura de imágenes

El detector de objetos funciona correctamente, puede dar resultados de varias imágenes a la vez, pero solo muestra en la ventana emergente aquella que tenga el nombre de 0.jpg, las etiquetas para las demás imágenes se muestran en la ventana de comandos de Anaconda. En este punto lo que hace el algoritmo es ubicarse en la carpeta, reconocer la imagen 0.jpg y llamarla para continuar con el resto del algoritmo, pero este proceso no concuerda muy bien con la definición de un sistema de visión artificial a pesar de ser construido con los conceptos básicos de éste. Para solucionarlo se opta por crear un tercer programa capaz de utilizar la cámara de un celular para capturar fotografías de la plántula y almacenarla en una carpeta con el nombre de 0.jpg y esperar a ser llamada por el programa de detección de objetos.

Para crear este programa es necesario descargar una aplicación al celular llamada DroidCam disponible en Google Play Store. Lo primero que se puede ver al abrir la aplicación es la IP Cam Access como se ve en la Figura 2.17, información imprescindible para la conexión de la computadora con el celular, pues puede variar para cada dispositivo.

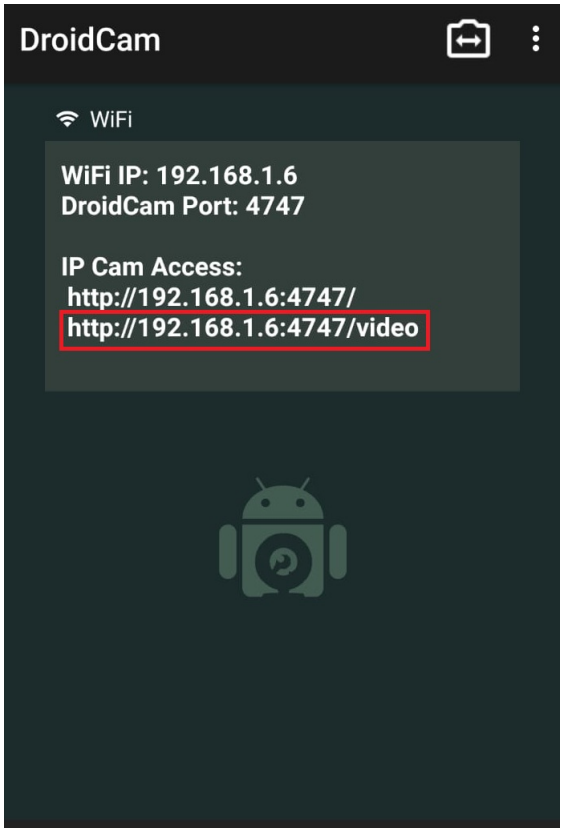


Figura 2.17: IP Cam Access de la aplicación DroidCam.

Conociendo este dato ya solo será cuestión de usarlo dentro del código que se encarga de conectar a la cámara del celular con el computador para poder capturar una imagen usando la tecla "Q", guardarla dentro de la carpeta con la que trabaja el detector de objetos y asignarle el nombre "0.jpg". El ciclo permite continuar tomando fotografías y nombrarlas numéricamente, sin embargo, es importante solo tomar una fotografía para utilizar correctamente el detector. Al ejecutar el programa el contador se reinicia por lo que si en el momento de guardar existe una imagen con el nombre que se necesite será reemplazada automáticamente.

Capítulo 3

Pruebas y Resultados

Para las evaluaciones tanto de la Red Neuronal como del detector de objetos se decide valorar los resultados con ayuda de una matriz de confusión, ésta compara las predicciones dadas por el método entrenado con la clase a la cual pertenecen [46] como se observa en la Figura 3.1.

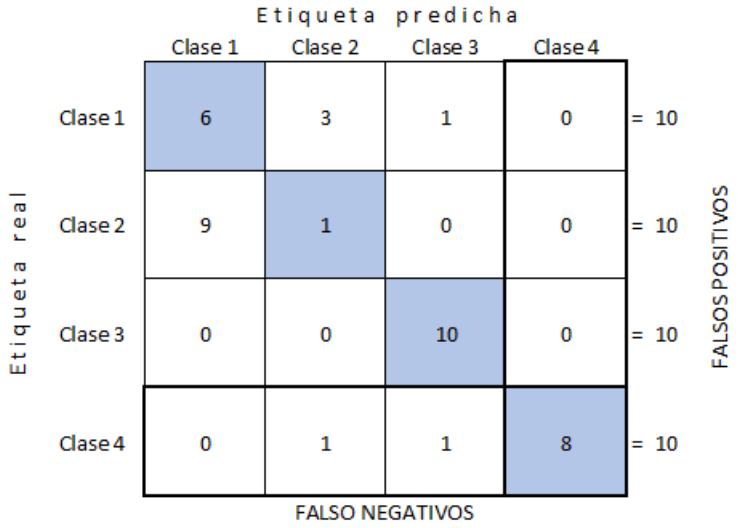


Figura 3.1: Ejemplo de una matriz de confusión.

A pesar de que la matriz no es una métrica por si sola, si es la base para calcular el desempeño del algoritmo, pero antes es necesario tener en cuenta algunas definiciones:

- Verdadero positivo (VP): cuando una plántula es reconocida correctamente por el algoritmo.
- Verdadero negativo (VN): cuando una plántula diferente a las 4 plántulas establecidas no es reconocida por el algoritmo.
- Falso negativo (FN): cuando una plántula diferente a las 4 plántulas establecidas es reconocida por el algoritmo.
- Falso positivo (FP): cuando una plántula es reconocida de manera incorrecta por el algoritmo.

En el presente proyecto se utiliza las siguientes ecuaciones para medir la funcionalidad del algoritmo:

- Tasa de detección: corresponde a la razón de los verdaderos positivos y la cantidad de imágenes utilizadas durante la evaluación del modelo.

$$\text{Tasa de detección} = \frac{\text{VP}}{\text{Número de datos}} \tag{3.1}$$

- Precisión: es la calidad que tiene las respuestas positivas de cada clase.

$$\text{Precisión} = \frac{\text{VP}}{(\text{VP} + \text{FP})} \tag{3.2}$$

- Recall: es la cantidad de datos que se colocan en etiquetas incorrectas.

$$\text{Recall} = \frac{\text{VP}}{(\text{VP} + \text{FN})} \tag{3.3}$$

- Exactitud: representa que tanto se aproxima el valor medido al valor real y utiliza todos los datos de la matriz.

$$\text{Exactitud} = \frac{\text{VP} + \text{VN}}{\text{VP} + \text{VN} + \text{FN} + \text{FP})} \tag{3.4}$$

3.1. Análisis de resultados de la Red Neuronal Convolucional

Debido a que en el entrenamiento de la red neuronal se ha utilizado imágenes que comparten un mismo fondo, tal y como se muestra en la Figura 2.3, los mejores resultados son utilizando un escenario lo más parecido posible, es decir, el espacio que rodea las plántulas debe ser en tonos marrones, negros y grises.

Una vez establecida la composición que la imagen debe cumplir para poder ser analizada por la CNN, se empieza a preparar el grupo de datos que van a participar en esta prueba. Estos datos corresponden a las imágenes reservadas en la carpeta "test", precisamente para esta etapa. Dicha carpeta forma parte de la base de datos descargada y contiene fotografías con el mismo fondo de las fotografías de las carpetas "train" y "validation". Ahora, con ayuda de un algoritmo extra se llaman todas las imágenes a ser clasificadas, el resultado se muestra en un archivo .csv. Para facilitar el proceso se evalúan 44 imágenes clasificadas por clase ya que de ser correcto las etiquetas deben ser iguales para todos los datos y aquellas que no lo sean representan los errores de la CNN. Esta información es descargada en un archivo .csv y es utilizada para la elaboración de la matriz de confusión de la Figura 3.2.

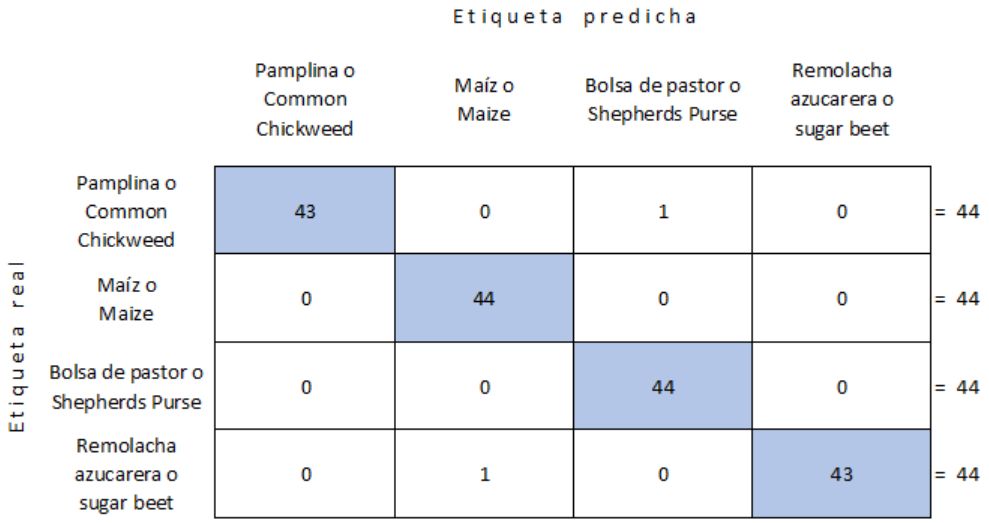


Figura 3.2: Matriz de confusión obtenida con los resultados de la evaluación del modelo entenado previamente.

Una vez creada la matriz de confusión se puede aplicar las ecuaciones 3.1, 3.2, 3.3 y 3.4, obteniendo:

	Pamplina o Common Chickweed	Maíz o Maize	Bolsa de pastor o Shepherds Purse	Remolacha azucarera o sugar beet
T=	0,97727273	1	1	0,977272727
P=	1	0,97777778	0,97777778	1
R=	0,97727273	1	1	0,977272727

Cuadro 3.1: Valores obtenidos en base a la matriz de confusión de los resultados obtenidos en la evaluación de la CNN.

Gracias a la matriz de confusión se puede evaluar los datos dividiéndolos por clase, siendo importante aclarar que los valores de precisión y recall están presentes debido a que en este proyecto es importante controlar los resultados FP Y FN, esto porque no se puede extraer una planta de cultivo confundiéndola como mala hierba, pero tampoco se puede mantener una mala hierba como parte del cultivo.

Para el caso de la clase 'Common Chickweed' la tasa de detección indica que de cada 100 imágenes que se analicen, solo 3 no serán detectadas. Con la precisión del 100 % se evidencia que todos los casos predichos correctamente resultaron ser casos positivos. Así también, gracias al recall se puede notar que el modelo predijo con éxito el 97 % de los positivos.

Con respecto a los datos obtenidos en la clase 'Maize' la tasa de detección indica que de cada 100 imágenes que se analicen, todas serán detectadas. Con la precisión del 97 %

se evidencia que todos los casos predichos correctamente resultaron ser casos positivos. Así también, gracias al recall se puede notar que el modelo predijo con éxito el 100 % de los positivos.

En relación con los datos obtenidos en la clase 'Shepherds Purse' la tasa de detección indica que de cada 100 imágenes que se analicen, todas serán detectadas. Con la precisión del 97 % se evidencia que todos los casos predichos correctamente resultaron ser casos positivos. Así también, gracias al recall se puede notar que el modelo predijo con éxito el 100 % de los positivos.

Para los datos obtenidos en la clase 'Sugar beet' la tasa de detección indica que de cada 100 imágenes que se analicen, solo 3 no serán detectadas. Con la precisión del 100 % se evidencia que todos los casos predichos correctamente resultaron ser casos positivos. Así también, gracias al recall se puede notar que el modelo predijo con éxito el 97 % de los positivos.

Con ayuda de la matriz también se puede obtener la exactitud del modelo en general, para lo cual se aplica la ecuación 3.4.

$$\text{Exactitud} = \frac{174}{176} \tag{3.5}$$

$$\text{Exactitud} = 98{,}9\,\% \tag{3.6}$$

El valor de la exactitud confirma el porcentaje de eficiencia obtenido durante el entrenamiento, siendo del 98.9 %.

Los resultados obtenidos durante la clasificación de los datos se aprecian de mejor manera en la Figura 3.3.

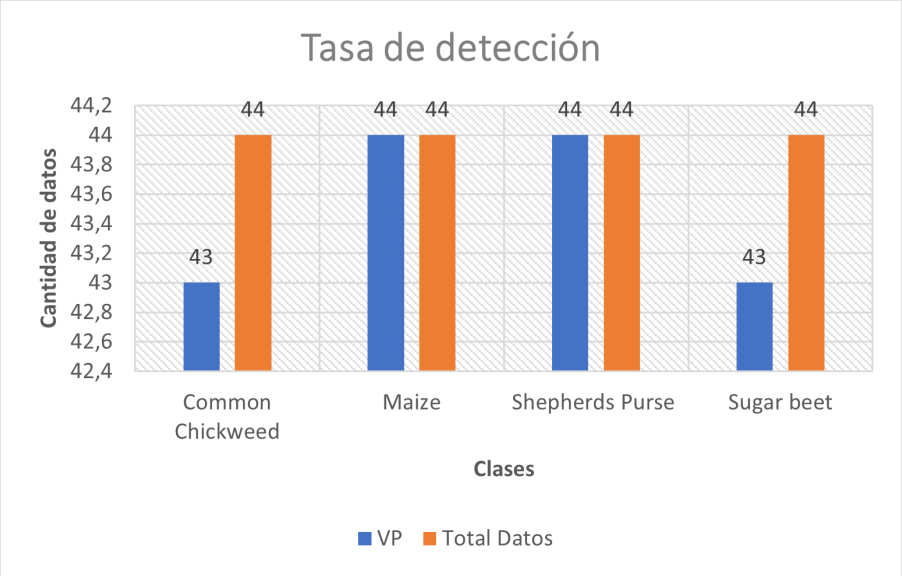


Figura 3.3: Número de aciertos del modelo de clasificación en comparación con el total de errores.

3.2. Análisis de resultados del detector de objetos

El diseño del detector de objetos se basa enteramente en el funcionamiento de la CNN, por lo que es normal que el algoritmo revele mejores resultados utilizando imágenes con las mismas condiciones, es decir, una plántula en un entorno de colores marrones, negros y grises. Con esta premisa se inician las pruebas del código, que, al funcionar de dos maneras distintas, como se indica en la sección 2.3.4, tienen diferentes escenarios.

3.2.1. Detector de objetos sin cámara

Nuevamente, con el fin de mantener la composición y el color de la imagen, los datos con los que se prueba el algoritmo se obtienen de la base de datos de manera aleatoria, agrupándose para cada clase 40 imágenes. Ya que la base de datos se compone de imágenes con formato .png se deben transformar al formato .jpg para pasar por el detector, algo que se hizo sin mayor dificultad utilizando Paint.

La prueba se realiza de manera individual, para cada imagen de cada clase, el algoritmo llama a la imagen de formato .jpg denominada "0" y después de ciertos segundos emite un resultado, el cual se registra manualmente en un documento Excel para luego contabilizar los resultados de acierto y falla.

Etiqueta real \ Etiqueta predicha	Pamplina o Common Chickweed	Maíz o Maize	Bolsa de pastor o Shepherds Purse	Remolacha azucarera o sugar beet	
Pamplina o Common Chickweed	40	0	0	0	= 40
Maíz o Maize	0	40	0	0	= 40
Bolsa de pastor o Shepherds Purse	0	0	40	0	= 40
Remolacha azucarera o sugar beet	0	0	0	40	= 40

Figura 3.4: Matriz de confusión con los resultados de la evaluación del detector de objetos sin cámara.

Una vez creada la matriz de confusión se puede aplicar las ecuaciones 3.1, 3.2, 3.3 y 3.4, obteniendo:

	Pamplina o Common Chickweed	Maíz o Maize	Bolsa de pastor o Shepherds Purse	Remolacha azucarera o sugar beet
VP=	1	1	1	1
P=	1	1	1	1
R=	1	1	1	1

Cuadro 3.2: Valores obtenidos en base a la matriz de confusión de los resultados obtenidos en la evaluación del detector de objetos sin cámara.

Como se muestra en el Cuadro 3.2 al aplicar se obtiene en todos los casos un resultado del 100 %.

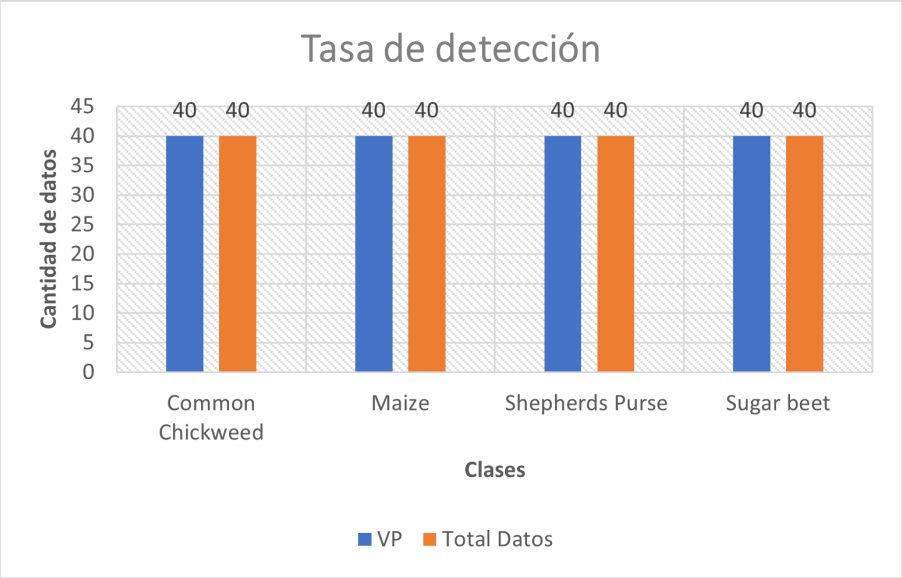


Figura 3.5: Número de aciertos del detector de objetos sin utilizar cámara en comparación con el total de datos utilizados para la prueba.

Como se muestra en la Figura 3.5 los resultados fueron realmente satisfactorios, con una precisión en el reconocimiento del 100 % pues el total de 160 datos no tuvieron ningún error. A pesar de los resultados, el sistema es ineficiente pues no trabaja con una cámara como debería hacerlo, sin embargo, esta información es valiosa ya que permite asegurar el correcto funcionamiento del detector de objetos en el escenario establecido.

3.2.2. Detector de objetos con cámara de celular

Lo ideal sería capturar imágenes de plántulas en un terreno, sin embargo, no existen las condiciones para crear un cultivo con las necesidades específicas que requiere este proyecto. Como alternativa se vuelve a hacer uso de la base de datos, de la cual se seleccionan de manera aleatoria 3 imágenes de cada clase para ser impresas y simular el

escenario. En una habitación iluminada solo por un foco central se ubica una superficie plana que con luz propia se ilumine en su totalidad, en este caso se usa la pantalla del computador encendida en luz blanca, en esta pantalla se coloca cada impresión con el fin de recibir la mayor cantidad de luz y facilitar el reconocimiento con la cámara del celular, el cual también debe capturar la fotografía en sentido horizontal y a una distancia prudente para cubrir la totalidad del área de la imagen impresa, siempre manteniendo una posición frontal, de lo contrario puede haber errores. Cada imagen impresa es capturada 10 veces para evaluar la exactitud del algoritmo. Las fotos son almacenadas en formato .jpg.

Con todas las fotografías capturadas, ya solo hace faltar utilizar el algoritmo detector de objetos para llamar individualmente cada fotografía, y al igual que la prueba anterior la denominación "0" no cambia. Finalmente, cada resultado se registra manualmente en un documento Excel para luego contabilizar la cantidad de aciertos y fallas.

	Common Chickweed	Maize	Shepherds Purse	Sugar beet	TOTAL
Common Chickweed 1			1	9	10
Common Chickweed 2	9	1			10
Common Chickweed 3	8		2		10
Maize 1		9		1	10
Maize 2		9	1		10
Maize 3		8		2	10
Shepherds Purse 1	4		5	1	10
Shepherds Purse 2		1	6	3	10
Shepherds Purse 3			10		10
Sugar beet 1				10	10
Sugar beet 2				10	10
Sugar beet 3				10	10

Cuadro 3.3: Etiquetas asignadas a cada grupo de imágenes.

Los resultados de los tres grupos se contabilizan en uno solo para poder armar la matriz de confusión de la Figura 3.6

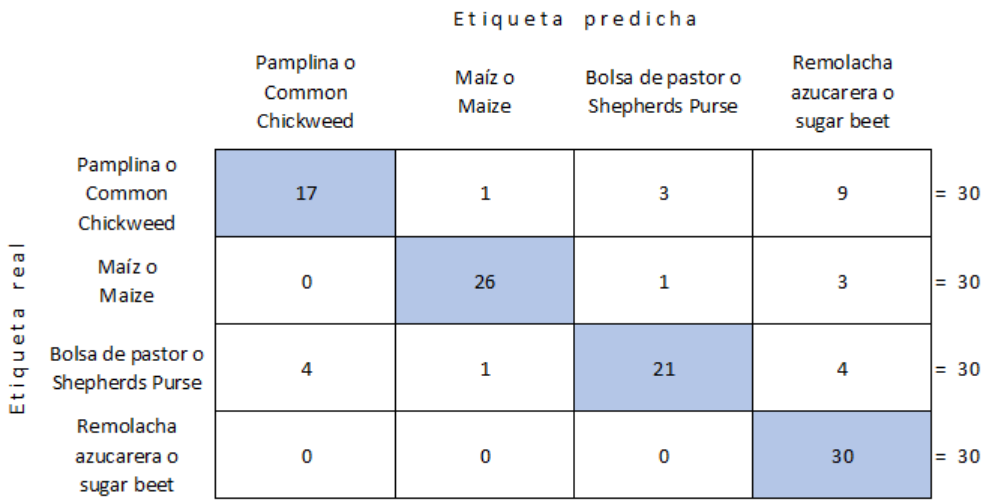


Figura 3.6: Matriz de confusión con los resultados de la evaluación del detector de objetos con cámara.

Una vez creada la matriz de confusión se puede aplicar las ecuaciones 3.1, 3.2, 3.3 y 3.4, obteniendo:

	Pamplina o Common Chickweed	Maíz o Maize	Bolsa de pastor o Shepherds Purse	Remolacha azucarera o sugar beet
VP=	0,56666667	0,86666667	0,7	1
P=	0,80952381	0,92857143	0,84	0,652173913
R=	0,56666667	0,86666667	0,7	1

Cuadro 3.4: Valores obtenidos en base a la matriz de confusión de los resultados obtenidos en la evaluación del detector de objetos con cámara.

Analizando el Cuadro 3.4, se puede ver que para el caso de la clase 'Common Chickweed' la tasa de detección indica que de cada 100 imágenes que se analicen, 44 no serán detectadas. Con la precisión se evidencia que el 80 % de los casos predichos correctamente resultaron ser casos positivos. Así también, gracias al recall se puede notar que el modelo predijo con éxito el 56 % de los positivos.

Con respecto a los datos obtenidos en la clase 'Maize' la tasa de detección indica que de cada 100 imágenes que se analicen, 14 no serán detectadas. Con la precisión se evidencia que el 92 % de los casos predichos correctamente resultaron ser casos positivos. Así también, gracias al recall se puede notar que el modelo predijo con éxito el 86 % de los positivos.

En relación con los datos obtenidos en la clase 'Shepherds Purse' la tasa de detección indica que de cada 100 imágenes que se analicen, 30 no serán detectadas. Con el valor de la precisión se evidencia que el 84 % de los casos predichos correctamente resultaron ser casos positivos. Así también, gracias al recall se puede notar que el modelo predijo con éxito el 70 % de los positivos.

Para los datos obtenidos en la clase 'Sugar beet' la tasa de detección indica que de cada 100 imágenes que se analicen, todas serán detectadas. Con el valor de la precisión se evidencia que el 65 % de los casos predichos correctamente resultaron ser casos positivos. Así también, gracias al recall se puede notar que el modelo predijo con éxito el 100 % de los positivos.

Con ayuda de la matriz también se obtiene el valor de la exactitud del modelo en general, para lo cual se aplica la ecuación 3.4.

$$\text{Exactitud} = \frac{94}{120} \tag{3.7}$$

$$\text{Exactitud} = 78{,}3\,\% \tag{3.8}$$

Ahora el porcentaje de exactitud en relación a la prueba anterior muestra un descenso, siendo ahora del 78,33 %, esto se debe a la selección de las imágenes y al ángulo de captura

de la foto. Pues como se ve en el Cuadro 3.3 para el grupo de imágenes de la etiqueta "Common Chickweed 1" la imagen seleccionada representa un error para el algoritmo, sin embargo, la mayoría de los resultados se muestra en una solo etiqueta, para los grupos de "Maize" y "Shepherde Purse" el error se debió a no mantener el mismo ángulo de captura establecido previamente, sino que hubo modificaciones para conocer los ángulos en el que mejor trabaja el algoritmo, pero también por la baja calidad de las imágenes. Finalmente, para el grupo "Sugar beet" se coincidió con imágenes que no son reconocidas como errores, pero también se procura mantener el celular en posición frontal a la impresión, sin subirlo ni bajarlo en lo más mínimo, obteniéndose un 100 % de precisión. Todos los datos con sus respectivos aciertos y errores se pueden apreciar de mejor manera en la Figura 3.7.

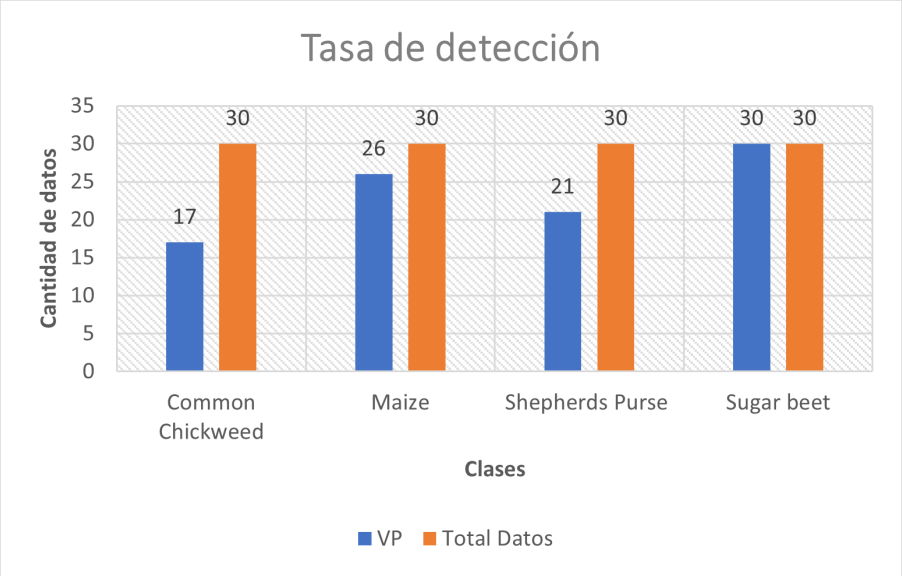


Figura 3.7: Número de aciertos del detector de objetos enlazado a la cámara del celular en comparación con el total de datos utilizados para la prueba.

Capítulo 4

Conclusiones y trabajo futuro

4.1. Conclusiones

- Como se observa en la Figura 2.4 la clase con menor número de imágenes contine un poco más de 40 registros, algo que no impidió crear un modelo eficiente.
- Al utilizar generadores para la creación de la CNN se tuvo la ventaja de que el entrenamiento y la validación se realizara en lotes por lo que no hubo la necesidad de cargar todos los datos en la memoria a la vez, algo que hubiese tomado una gran cantidad de tiempo ya que la cantidad de imágenes sobrepasa los 2 000 datos.
- Como se ve en la ecuación 3.8, la precisión del sistema disminuye en comparación al detector de objetos sin cámara, esto puede deberse a que a pesar de que en cada captura se intentó mantener las mismas condiciones, hubo pequeñas variaciones que modificaron el resultado, como el ángulo de la captura, la calidad de las imágenes y la iluminación. Se debe tomar en cuenta que en las fallas tendían a etiquetarse en su mayoría a 'Sugar beet' una clase que no tuvo problemas en la clasificación.
- Es importante proyectos de este tipo, que analicen plantas en su fase de plántula ya que permite al agricultor gastar recursos en plantas no deseadas y también permite analizar las plantas, cada uno en su propio espacio pues al crecer es común que las hojas de las plantas estén sobrepuestas entre ellas lo que es contraproducente en el reconocimiento.

4.2. Trabajo futuro

En el presente proyecto se desarrolló un algoritmo que clasifica 4 tipos de plántulas, las cuales son: pamplina, maíz, bolsa de pastor y remolacha azucarera. Con el fin de mejorar y explotar el estudio se propone como trabajo futuro lo siguiente:

- Como ya se conoce, el modelo RGB presenta desventajas en el uso de detección de objetos debido a las superficies sombreadas, por lo que podría realizarse el entrenamiento de la CNN con imágenes de representación HSI.
- Como se observó en la sección de resultados[1], el algoritmo funciona correctamente en un entorno controlado, por lo que para crear uno de funcionamiento que abarque

[1] Véase en la sección 3.1

mayores escenarios, el cambio debe iniciar, si es posible desde la base de datos, pero si no, puede modificarse la técnica de procesamiento de imágenes[2] y volver a entrenar la CNN.

- Aprovechar que el algoritmo puede modificarse para trabajar con la cámara de cualquier celular para trabajar con una cámara que capture imágenes de mejor calidad y así verificar la idea de cambiar el escenario de funcionamiento.
- Sería interesante modificar el código del detector de objetos que utiliza un programa para el uso de la cámara del celular para que pueda identificar la plántula en tiempo real y no esperar capturar la imagen.

[2]Véase la sección 2.2.1

Bibliografía

[1] Belal AM Ashqar, Bassem S Abu-Nasser y Samy S Abu-Naser. «Plant seedlings classification using deep learning». En: (2019).

[2] Ece Olcay Güneş y Sercan Aygün. «Growth monitoring of plants using active contour technique». En: *2017 6th International Conference on Agro-Geoinformatics*. IEEE. 2017, págs. 1-5.

[3] Martín Montalvo Martínez. «Técnicas de visión artificial para la segmentación y detección de líneas de cultivo en imágenes agrícolas». En: (2015).

[4] Rogger David Gaviria Montoya, Cristhian David Marin Hurtado y col. «Sistema de inspección y clasificación de hojas de plantas medicinales por medio de visión artificial». En: (2018).

[5] Juan Francisco Sotomayor, Alejandro Paúl Gómez y Andrés Fernando Cela. «Sistema de visión artificial para el análisis de imágenes de cultivo basado en texturas orientadas». En: *Revista Politécnica* 33.1 (2014).

[6] Jesús Piña Muñoz. «Sistema de automatización de reconocimiento de malas hierbas en cultivos de algodón basado en aprendizaje». En: (2018).

[7] Gamal ElMasry, Nasser Mandour, Marie-Hélène Wagner, Didier Demilly, Jerome Verdier, Etienne Belin y David Rousseau. «Utilization of computer vision and multispectral imaging techniques for classification of cowpea (Vigna unguiculata) seeds». En: *Plant methods* 15.1 (2019), págs. 1-16.

[8] Zhang Xiao, Yu Tan, Xingxing Liu y Shenghui Yang. «Classification method of plug seedlings based on transfer learning». En: *Applied Sciences* 9.13 (2019), pág. 2725.

[9] Eddie Ángel Sobrado Malpartida. «Sistema de visión artificial para el reconocimiento y manipulación de objetos utilizando un brazo robot». En: (2003).

[10] Jefferson Geovanny Bailón Lucas y Marcos Antonio Rodríguez Vera. «Desarrollo de un sistema reconocedor de frutas para supermercados aplicando visión artificial». B.S. thesis. Universidad de Guayaquil Facultad de Ciencias Matemáticas y Físicas Carrera, 2019.

[11] A. Narayanan, E. Shi y B. I. P. Rubinstein. «Link prediction by de-anonymization: How We Won the Kaggle Social Network Challenge». En: *The 2011 International Joint Conference on Neural Networks*. 2011, págs. 1825-1834. DOI: 10.1109/IJCNN.2011.6033446.

[12] *Plant Seedlings Classification Kaggle*. https://www.kaggle.com/c/plant-seedlings-classification/data?select=test. (Accessed on 03/17/2021).

[13] *Seedlings - Pretrained keras models | Kaggle*. URL: https://www.kaggle.com/gaborfodor/seedlings-pretrained-keras-models.

[14] Laura Crook. *Black-grass: the farmer's nemesis « Botany One*. https://www.botany.one/2017/04/black-grass-farmers-nemesis/. (Accessed on 04/11/2021). 2017.

[15] Acuacultura y Pesca Ministerio de Agricultura Ganadería. *13_1939_00_s.pdf*. https://members.wto.org/crnattachments/2013/sps/ECU/13_1939_00_s.pdf. (Accessed on 04/11/2021). 2013.

[16] The Editors of Encyclopaedia Britannica. *Charlock | plant | Britannica*. https://www.britannica.com/plant/charlock. (Accessed on 04/11/2021). 2018.

[17] Garden Organic. *Charlock | www.gardenorganic.org.uk*. https://www.gardenorganic.org.uk/weeds/charlock. (Accessed on 04/11/2021). 2007.

[18] *Cleavers - Massey University*. https://www.massey.ac.nz/massey/learning/colleges/college-of-sciences/clinics-and-services/weeds-database/cleavers.cfm. (Accessed on 04/11/2021).

[19] Eric Orr. *Chickweed: Edible and Delicious Weed*. https://www.wildedible.com/chickweed. (Accessed on 04/11/2021).

[20] *What Are Essential Oils, and Do They Work?* https://www.healthline.com/nutrition/what-are-essential-oils#how-they-work. (Accessed on 04/11/2021).

[21] Wilmer Xavier Gómez Yujato. *Identificiación de arvenses presentes en el cultivo de cacao (Theobroma cacao L.) en Montalvo, Vinces y Urdaneta*. shorturl.at/gpFM1. (Accessed on 04/11/2021). 2016.

[22] Universidad Pública de Navarra. *Chenopodium album L.* https://www.unavarra.es/herbario/htm/Chen_albu.htm. (Accessed on 04/12/2021).

[23] *Chenopodium album, Cenizo*. https://www.asturnatura.com/especie/chenopodium-album.html. (Accessed on 04/12/2021).

[24] Bayer. *Loose silky-bent*. https://cropscience.bayer.co.uk/threats/grass-weeds/loose-silky-bent/. (Accessed on 04/12/2021).

[25] University of Hertfordshire. *Weeds - Grass Weeds Information and Photo Gallery - Loose silky-bent*. http://adlib.everysite.co.uk/adlib/defra/content.aspx?id=000HK277ZX.0AGZCS09TISKYO. (Accessed on 04/12/2021). 2011.

[26] Agrobase. *Loose silky-bent*. https://agrobaseapp.com/ireland/weed/loose-silky-bent-1. (Accessed on 04/12/2021).

[27] Julia Máxima Uriarte. *Maíz: historia, cultivo, variedades, usos y características*. https://www.caracteristicas.co/maiz/. (Accessed on 04/12/2021). 2020.

[28] Nature Spot. *Scentless Mayweed | NatureSpot*. https://www.naturespot.org.uk/species/scentless-mayweed. (Accessed on 04/12/2021).

[29] *Capsella bursa-pastoris - ficha informativa*. http://www.conabio.gob.mx/malezasdemexico/brassicaceae/capsella-bursa-pastoris/fichas/ficha.htm. (Accessed on 04/12/2021).

[30] Valeria Nataly Villota Cevallos. *Evaluación de tres tipos de coberturas vegetales (rastrojos de cultivos de cebada, maíz y arveja) para evitar la propagación de malezas en el cultivo de papa (Solanum tuberosum) en el cantón Huaca - Carchi, Ecuador*. shorturl.at/mqxM2. (Accessed on 04/12/2021). 2017.

[31] Darwin Foundation. *Capsella bursa-pastoris*. shorturl.at/yEP25. (Accessed on 04/12/2021).

[32] *Small-flowered Cranesbill, Geranium pusillum - Flowers - NatureGate*. https://www.luontoportti.com/suomi/en/kukkakasvit/small-flowered-cranesbill. (Accessed on 04/12/2021).

[33] *Ecuador Remolacha Azucarera, 1961-2020 - knoema.com*. shorturl.at/qADF6. (Accessed on 04/12/2021). 2019.

[34] Carlos Avilés Cruz Juan Villegas Cortéz Fidel López Saca Andrés Ferreyra Ramírez. *Red neuronal convolucional con extracción de características multi-columna para clasificación de imágenes*. shorturl.at/brASY. (Accessed on 04/14/2021). 2019.

[35] Elia Pérez Pérez. *Diseño de una metodología para el procesamiento de imágenes mamográficas basada en técnicas de aprendizaje profundo*. http://oa.upm.es/47315/1/PFC_ELIA_PEREZ_PEREZ_2017.pdf. (Accessed on 04/14/2021). 2017.

[36] *Seedling Image Classification Using Keras | Kaggle*. https://www.kaggle.com/reedr1208/seedling-image-classification-using-keras. (Accessed on 03/16/2021).

[37] Nicolás Aguirre. *Procesamiento de imágenes*. shorturl.at/eqLO8. (Accessed on 04/12/2021).

[38] F. Ortiz P. Gil F. Torres. «Detección de objetos por segmentación multinivel combinada de espacios de color». En.

[39] G. Ambrosio V. Arévalo J. González. *La librería de visión artificial OpenCV aplicación a la docencia e investigación*. http://mapir.isa.uma.es/varevalo/drafts/arevalo2004lva1.pdf. (Accessed on 04/13/2021).

[40] Manpreet Singh Minhas. *Image Data Generators in Keras. How to effectively and efficiently use data generators in Keras for Computer Vision applications of Deep Learning*. https://towardsdatascience.com/image-data-generators-in-keras-7c5fc6928400. (Accessed on 04/13/2021). 2019.

[41] *Keras: the Python deep learning API*. shorturl.at/ahmxS. (Accessed on 04/13/2021).

[42] Sara Domínguez Pavón. *Identificación del modelo de cámara mediante Redes Neuronales Convolucionales*. http://bibing.us.es/proyectos/abreproy/71426/fichero/TFM-1426-DOMINGUEZ.pdf. (Accessed on 04/14/2021). 2019.

[43] Eugenio García Sánchez. *Introducción a las redes neuronales de convolución. Aplicación a la visión por ordenador*. https://core.ac.uk/download/pdf/290002463.pdf. (Accessed on 04/14/2021). 2019.

[44] Jason Brownlee. *Difference Between a Batch and an Epoch in a Neural Network*. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/. (Accessed on 04/14/2021). 2019.

[45] *Deep Learning for Computer Vision with Python*. 2018.

[46] Diego Andres Acuña Escobar. *Visión artificial aplicada a la detección e identificación de personas en tiempo real*. https://bibdigital.epn.edu.ec/bitstream/15000/20098/1/CD-9539.pdf. (Accessed on 06/10/2021). 2018.

Apéndice A

Algoritmo

El código que se muestra en esta sección fue diseñado en una herramienta de Python, denominado Jupyter Notebook.

A.1. Algoritmo de la CNN que clasifica 4 tipos de plántulas

A.1.1. Librerías utilizadas

```
In [2]: import pandas as pd
        import numpy as np
        import keras
        import os
        import matplotlib.pyplot as plt
        from keras.preprocessing import image
        import random
        import pickle
        from keras import models, layers, callbacks
        import shutil
        import cv2
        from math import sqrt, floor
        from prettytable import PrettyTable

        from mpl_toolkits.mplot3d import Axes3D
        from matplotlib import cm
        from matplotlib import colors

        import csv
```

A.1.2. Ingreso a la base de datos a utilizar

```
In [2]: #move to directory containing data
        os.chdir('C:/Users/Jhenifer)
```

In [4]:

```
def print_bold(text):
    print('\033[1m{}\033[0m'.format(text))

def list_files(startpath):
    for root, dirs, files in os.walk(startpath):
        level = root.replace(startpath, '').count(os.sep)
        indent = ' ' * 4 * (level)

        dir_name= '{}{}/'.format(indent, os.path.basename(root))
        if dir_name.strip().startswith('.'):
            continue

        print_bold('\n'+dir_name)

        subindent = ' ' * 4 * (level + 1)
        if level==0:
            for f in files:
                if f.startswith('.'):
                    continue
                print('{}{}'.format(subindent, f))
        else:
            #if len(files)>0:
            #   print('{}File Count: {}'.format(subindent, len(files)))
            for i, f in enumerate(files):
                print('{}{}'.format(subindent, f))
                if i==2:
                    print('{}{}'.format(subindent, '...'))
                    break

list_files(os.getcwd())
```

```
4DataBase/
    4seedling.ipynb

    test/
        0021e90e4.png
        003d61042.png
        007b3da8b.png
        ...

    train/

        Common Chickweed/
            00b6eee9f.png
            00ba5f88a.png
            00d33935c.png
            ...

        Maize/
            006196e1c.png
            0086c28b2.png
            00a18f05e.png
            ...

        Shepherds Purse/
            006a4d00d.png
            00dd0d16a.png
            01aef64d2.png
            ...

        Sugar beet/
            0026b7a30.png
            00626e3be.png
            008753052.png
            ...
```

A.1.3. Extracción de la cantidad de datos dentro de la carpeta "train"

In [104]:
```
classes= []
sample_counts= []

for f in os.listdir('train'):
    train_class_path= os.path.join('train', f)
    if os.path.isdir(train_class_path):
        classes.append(f)
        sample_counts.append(len(os.listdir(train_class_path)))

plt.rcdefaults()
fig, ax = plt.subplots()

# Example data
y_pos = np.arange(len(classes))

ax.barh(y_pos, sample_counts, align='center')
ax.set_yticks(y_pos)
ax.set_yticklabels(classes)
ax.invert_yaxis()  # labels read top-to-bottom
ax.set_xlabel('Sample Counts')
ax.set_title('Sample Counts Per Class')

plt.show()
```

A.1.4. Gráfica de cuatro imágenes aleatorias de cada clase dentro de la carpeta "train"

In [57]:
```
fig= plt.figure(figsize= (10, 15))
fig.suptitle('Random Samples From Each Class', fontsize=14, y=.92, horizontalalignment='center', weight='bold')

columns = 5
rows = 4
for i in range(4):
    sample_class= os.path.join('train',classes[i])
    for j in range(1,6):
        fig.add_subplot(rows, columns, i*5+j)
        plt.axis('off')
        if j==1:
            plt.text(0.0, 0.5,str(classes[i]).replace(' ','\n'), fontsize=13, wrap=True)
            continue

        random_image= os.path.join(sample_class, random.choice(os.listdir(sample_class)))
        #from keras.preprocessing.image
        img = image.load_img(random_image, target_size=(150, 150))
        img= image.img_to_array(img)
        img/=255.
        plt.imshow(img)

plt.show()
```

A.1.5. Creación de la carpeta “validation”

In [58]:
```
#create validation set
def create_validation(validation_split=0.2):
    if os.path.isdir('validation'):
        print('Validation directory already created!')
        print('Process Terminated')
        return
    os.mkdir('validation')
    for f in os.listdir('train'):
        train_class_path= os.path.join('train', f)
        if os.path.isdir(train_class_path):
            validation_class_path= os.path.join('validation', f)
            os.mkdir(validation_class_path)
            files_to_move= int(0.2*len(os.listdir(train_class_path)))

            for i in range(files_to_move):
                random_image= os.path.join(train_class_path, random.choice(os.listdir(train_class_path)))
                shutil.move(random_image, validation_class_path)
    print('Validation set created successfully using {:.2%} of training data'.format(validation_split))
```

In [59]:
```
create_validation()
```

```
Validation directory already created!
Process Terminated
```

A.1.6. Comparación de la cantidad de datos dentro de la carpeta “train” y “validation”

In [103]:
```
sample_counts= {}

for i, d in enumerate(['train', 'validation']):

    classes= []
    sample_counts[d]= []

    for f in os.listdir(d):
        train_class_path= os.path.join(d, f)
        if os.path.isdir(train_class_path):
            classes.append(f)
            sample_counts[d].append(len(os.listdir(train_class_path)))

    #fig, ax= plt.subplot(221+i)
    fig, ax = plt.subplots()

    # Example data
    y_pos = np.arange(len(classes))

    ax.barh(y_pos, sample_counts[d], align='center')
    ax.set_yticks(y_pos)
    ax.set_yticklabels(classes)
    ax.invert_yaxis()  # labels read top-to-bottom
    ax.set_xlabel('Sample Counts')
    ax.set_title('{} Sample Counts Per Class'.format(d.capitalize()))

plt.show()
```

A.1.7. Análisis de color

Muestra aleatoria de 50 píxeles de 10 imágenes de cada clase

In [61]:
```
def pull_random_pixels(samples_per_class, pixels_per_sample):
    total_pixels= 4*samples_per_class*pixels_per_sample
    random_pixels= np.zeros((total_pixels, 3), dtype=np.uint8)
    for i in range(4):
        sample_class= os.path.join('train',classes[i])
        for j in range(samples_per_class):

            random_image= os.path.join(sample_class, random.choice(os.listdir(sample_class)))
            img= cv2.imread(random_image)
            img= cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img=np.reshape(img, (img.shape[0]*img.shape[1], 3))
            new_pixels= img[np.random.randint(0, img.shape[0], pixels_per_sample)]

            start_index=pixels_per_sample*(i*samples_per_class+j)
            random_pixels[start_index:start_index+pixels_per_sample,:]= new_pixels

    h= floor(sqrt(total_pixels))
    w= total_pixels//h

    random_pixels= random_pixels[np.random.choice(total_pixels, h*w, replace=False)]
    random_pixels= np.reshape(random_pixels, (h, w, 3))
    return random_pixels

random_pixels= pull_random_pixels(10, 50)

plt.figure()
plt.suptitle('Random Samples From Each Class', fontsize=14, horizontalalignment='center')
plt.imshow(random_pixels)
plt.show()
```

Análisis de color dentro del modelo RGB

In [62]:
```
r, g, b = cv2.split(random_pixels)
fig = plt.figure(figsize=(8, 8))
axis = fig.add_subplot(1, 1, 1, projection="3d")
axis.view_init(20, 120)

pixel_colors = random_pixels.reshape((np.shape(random_pixels)[0]*np.shape(random_pixels)[1], 3))
norm = colors.Normalize(vmin=-1.,vmax=1.)
norm.autoscale(pixel_colors)
pixel_colors = norm(pixel_colors).tolist()


axis.scatter(r.flatten(), g.flatten(), b.flatten(), facecolors=pixel_colors, marker=".")
axis.set_xlabel("Red")
axis.set_ylabel("Green")
axis.set_zlabel("Blue")
plt.show()
```

Análisis de color dentro del modelo HSV

In [63]:
```
hsv_img = cv2.cvtColor(np.uint8(random_pixels), cv2.COLOR_RGB2HSV)

h, s, v = cv2.split(hsv_img)
fig = plt.figure(figsize=(8,8))
axis = fig.add_subplot(1, 1, 1, projection="3d")
axis.view_init(50, 240)

axis.scatter(h.flatten(), s.flatten(), v.flatten(), facecolors=pixel_colors, marker=".")
axis.set_xlabel("Hue")
axis.set_ylabel("Saturation")
axis.set_zlabel("Value")
plt.show()
```

Análisis de color dentro del modelo HS

In [21]:
```
hsv_img = cv2.cvtColor(np.uint8(random_pixels), cv2.COLOR_RGB2HSV)

h, s, v = cv2.split(hsv_img)
fig = plt.figure(figsize=(6,6))
axis = fig.add_subplot(1, 1, 1)

axis.scatter(h.flatten(), s.flatten(), facecolors=pixel_colors, marker=".")
axis.set_xlabel("Hue")
axis.set_ylabel("Saturation")
plt.show()
```

Extracción del fondo de las imágenes

In [64]:
```
lower_bound= (24, 50, 0)
upper_bound= (55, 255, 255)

fig= plt.figure(figsize=(10, 10))
fig.suptitle('Random Pre-Processed Image From Each Class', fontsize=14, y=.92,
             horizontalalignment='center', weight='bold')

for i in range(4):
    sample_class=os.path.join('train',classes[i])
    random_image= os.path.join(sample_class, random.choice(os.listdir(sample_class)))
    img= cv2.imread(random_image)
    img= cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img= cv2.resize(img, (150, 150))

    hsv_img= cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    mask = cv2.inRange(hsv_img, lower_bound, upper_bound)
    result = cv2.bitwise_and(img, img, mask=mask)

    fig.add_subplot(6, 4, i*2+1)
    plt.imshow(img)
    plt.axis('off')

    fig.add_subplot(6, 4, i*2+2)
    plt.imshow(result)
    plt.axis('off')

plt.show()
```

Función ImageDataGenerator

In [26]:
```
def color_segment_function(img_array):
    img_array= np.rint(img_array)
    img_array= img_array.astype('uint8')
    hsv_img= cv2.cvtColor(img_array, cv2.COLOR_RGB2HSV)
    mask = cv2.inRange(hsv_img, (24, 50, 0), (55, 255, 255))
    result = cv2.bitwise_and(img_array, img_array, mask=mask)
    result= result.astype('float64')
    return result
```

In [27]:
```
#files moved from "test/" to "test/test_images" since ImageDataGenerator requires
#files to be within a subfolder
os.rename('test', 'test_images')
shutil.move('test_images', 'test/test_images')
```

Out[27]:
```
'test/test_images'
```

In [65]:
```
#image function from keras.preprocessing
train_datagen = image.ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.0,
      height_shift_range=0.0,
      shear_range=0.0,
      zoom_range=0.0,
      horizontal_flip=True,
      vertical_flip= True,
      preprocessing_function=color_segment_function,
      fill_mode='nearest')

test_datagen = image.ImageDataGenerator(rescale=1./255,
           preprocessing_function=color_segment_function)
```

In [66]:
```
train_generator = train_datagen.flow_from_directory(
  'train',
  target_size=(150, 150),
  batch_size=16,
  class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        'validation',
        target_size=(150, 150),
        batch_size=16,
        class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
        'test',
        target_size=(150,150),
        batch_size=1,
        class_mode='categorical',
        shuffle=False)
```

```
Found 1159 images belonging to 4 classes.
Found 289 images belonging to 4 classes.
Found 794 images belonging to 1 classes.
```

A.1.8. Cálculo de pesos para cada clase

In [82]:
```
#get class indices and labels. calculate class weight
label_map = {}
for k, v in train_generator.class_indices.items():
    label_map[v]=k

class_counts= pd.Series(train_generator.classes).value_counts()
class_weight= {}

for i, c in class_counts.items():
    class_weight[i]= 1.0/c

norm_factor= np.mean(list(class_weight.values()))

for k in class_counts.keys():
    class_weight[k]= class_weight[k]/norm_factor

t = PrettyTable(['class_index', 'class_label', 'class_weight'])
for i in sorted(class_weight.keys()):
    t.add_row([i, label_map[i], '{:.2f}'.format(class_weight[i])])
print(t)
```

A.1.9. Estructura de la CNN

In [79]:
```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), input_shape=(150, 150, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.1))


model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.1))



model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.1))


model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.1))


model.add(layers.Flatten())
model.add(layers.Dropout(0.4))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.4))

model.add(layers.Dense(4, activation='softmax'))
```

In [80]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
dropout_18 (Dropout)         (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
dropout_19 (Dropout)         (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
dropout_20 (Dropout)         (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
dropout_21 (Dropout)         (None, 7, 7, 128)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 6272)              0
_________________________________________________________________
dropout_22 (Dropout)         (None, 6272)              0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               1605888
_________________________________________________________________
dropout_23 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 4)                 1028
=================================================================
Total params: 1,847,748
Trainable params: 1,847,748
Non-trainable params: 0
_________________________________________________________________
```

A.1.10. Entrenamiento

```
In [96]: best_cb= callbacks.ModelCheckpoint('model_best_1.h5',
                                            monitor='val_accuracy',
                                            verbose=1,
                                            save_best_only=True,
                                            save_weights_only=False,
                                            mode='auto',
                                            period=1)

         opt= keras.optimizers.Adam(lr=0.0005, amsgrad=True)
```

In [97]:
```
model.compile(optimizer=opt,
                loss='categorical_crossentropy',
                metrics=['accuracy'])

history = model.fit_generator(
                    train_generator,
                    class_weight= class_weight,
                    steps_per_epoch= 72,
                    epochs=100,
                    validation_data=validation_generator,
                    validation_steps= 18,
                    verbose=1,
                    callbacks= [best_cb])
```

In [4]:
```
#Load best model from training
model= models.load_model('model_best_1.h5')
print(model)
```

A.1.11. Pruebas de reconocimiento

In [100]:
```
pred= model.predict_generator(test_generator, steps= test_generator.n, verbose=1)
```

```
794/794 [==============================] - 14s 17ms/step
```

In [101]:
```
predicted_class_indices=np.argmax(pred,axis=1)

prediction_labels = [label_map[k] for k in predicted_class_indices]
filenames= test_generator.filenames
```

In [102]:
```
csvfile= open('ray_reed_submission.csv', 'w', newline='')
writer= csv.writer(csvfile)

headers= ['file', 'species']

writer.writerow(headers)
t = PrettyTable(headers)
for i, f, p in zip(range(len(filenames)), filenames, prediction_labels):
    writer.writerow([os.path.basename(f),p])
    if i <10:
        t.add_row([os.path.basename(f), p])
    elif i<13:
        t.add_row(['.', '.'])
csvfile.close()
print(t)
```

```
+---------------+------------------+
|      file     |     species      |
+---------------+------------------+
| 0021e90e4.png | Common Chickweed |
| 003d61042.png |    Sugar beet    |
| 007b3da8b.png |    Sugar beet    |
| 0086a6340.png | Common Chickweed |
| 00c47e980.png |    Sugar beet    |
| 00d090cde.png | Common Chickweed |
| 00ef713a8.png | Common Chickweed |
| 01291174f.png |    Sugar beet    |
| 026716f9b.png | Common Chickweed |
| 02cfeb38d.png | Common Chickweed |
|       .       |         .        |
|       .       |         .        |
|       .       |         .        |
+---------------+------------------+
```

A.2. Algoritmo del detector de objetos basado en la CNN

A.2.1. Librerías utilizadas

```
# import the necessary packages
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications import densenet
from tensorflow.keras.applications import vgg16
from tensorflow.keras.applications import efficientnet
from keras import models, layers, callbacks
from tensorflow.keras.applications.resnet import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications import imagenet_utils
from imutils.object_detection import non_max_suppression
from pyimagesearch.detection_helpers import sliding_window
from pyimagesearch.detection_helpers import image_pyramid
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np
import argparse
import imutils
import time
import os
import cv2
import pandas as pd
```

A.2.2. Condiciones iniciales

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=False, default="ImaGenerator/images/0.jpg",
	help="path to the input image")
ap.add_argument("-s", "--size", type=str, default="(100, 100)",
	help="ROI size (in pixels)")
ap.add_argument("-c", "--min-conf", type=float, default=0.9,
	help="minimum probability to filter weak detections")
ap.add_argument("-v", "--visualize", type=int, default=-1,
	help="whether or not to show extra visualizations for debugging")
args = vars(ap.parse_args())
print(args)

# initialize variables used for the object detection procedure
WIDTH = 150
PYR_SCALE = 1.5
WIN_STEP = 16
ROI_SIZE = eval(args["size"])
INPUT_SIZE = (150, 150)
```

A.2.3. Carga del modelo CNN

```
# load the input image from disk, resize it such that it has the
# has the supplied width, and then grab its dimensions
orig = cv2.imread(args["image"])
print("first",orig)
orig = imutils.resize(orig, width=WIDTH)
print("second",orig)

(H, W) = orig.shape[:2]
print(H,W)
```

A.2.4. Pirámides de la imagen

```
# initialize two lists, one to hold the ROIs generated from the image
# pyramid and sliding window, and another list used to store the
# (x, y)-coordinates of where the ROI was in the original image
rois = []
locs = []

print("rois=",rois)
print("locs=",locs)

# loop over the image pyramid
for image in pyramid:
    # determine the scale factor between the *original* image
    # dimensions and the *current* layer of the pyramid
    scale = W / float(image.shape[1])
    print("scale",scale)
#    print(W)
#    print(float(image.shape[1]))
#    print(image.shape[1])

    # for each layer of the image pyramid, loop over the sliding
    # window locations
    for (x, y, roiOrig) in sliding_window(image, WIN_STEP, ROI_SIZE):
        # scale the (x, y)-coordinates of the ROI with respect to the
        # *original* image dimensions
        x = int(x * scale)
        y = int(y * scale)
        w = int(ROI_SIZE[0] * scale)
        h = int(ROI_SIZE[1] * scale)

        # take the ROI and pre-process it so we can later classify
        # the region using Keras/TensorFlow
        roi = cv2.resize(roiOrig, INPUT_SIZE)
        roi = img_to_array(roi)
        roi = preprocess_input(roi)

        # update our list of ROIs and associated coordinates
        rois.append(roi)
        locs.append((x, y, x + w, y + h))
        # check to see if we are visualizing each of the sliding
        # windows in the image pyramid
        if 1> 0:
            # clone the original image and then draw a bounding box
            # surrounding the current region
            clone = orig.copy()
            cv2.rectangle(clone, (x, y), (x + w, y + h),(0, 255, 0), 2)

            # show the visualization and current ROI
            cv2.imshow("Visualization", clone)
            cv2.imshow("ROI", roiOrig)
            cv2.waitKey(0)
```

A.2.5. Ventanas deslizantes

```
# classify each of the proposal ROIs using ResNet and then show how
# long the classifications took
print("[INFO] classifying ROIs...")

#move to directory containing data
os.chdir('C:/Users/Jhenifer/Desktop/DetectionON/ImaGenerator')

def color_segment_function(img_array):
    img_array= np.rint(img_array)
    img_array= img_array.astype('uint8')
```

```
    hsv_img= cv2.cvtColor(img_array, cv2.COLOR_RGB2HSV)
    mask = cv2.inRange(hsv_img, (24, 50, 0), (55, 255, 255))
    result = cv2.bitwise_and(img_array, img_array, mask=mask)
    result= result.astype('float64')
    return result

from tensorflow.keras.preprocessing import image

test_datagen = image.ImageDataGenerator(rescale=1./255, preprocessing_function=color_segment_function)

#move to directory containing data
os.chdir('C:/Users/Jhenifer/Desktop/DetectionON')

test_generator = test_datagen.flow_from_directory(
        'ImaGenerator',
        target_size=(150,150),
        batch_size=1,
        class_mode='categorical',
        shuffle=False)

preds= model.predict_generator(test_generator, steps= test_generator.n, verbose=1)
print(preds)

# decode the predictions and initialize a dictionary which maps class
# labels (keys) to any ROIs associated with that label (values)

#move to directory containing data
os.chdir('C:/Users/Jhenifer/Desktop/DetectionON')
import json

def decode_predictions(preds, top=4, class_list_path=None):
    if len(preds.shape) != 2 or preds.shape[1] != 4:
        raise ValueError('`decode_predictions` expects '
                         'a batch of predictions '
                         '(i.e. a 2D array of shape (samples, 1000)). '
                         'Found array with shape: ' + str(preds.shape))
    CLASS_INDEX = json.load(open(class_list_path))
    results = []
    for pred in preds:
        top_indices = pred.argsort()[-top:][::-1]
        result = [tuple(CLASS_INDEX[str(i)]) + (pred[i],) for i in top_indices]
        result.sort(key=lambda x: x[2], reverse=True)
        results.append(result)
    return results

preds = decode_predictions(preds, top=1, class_list_path=
            'C:/Users/Jhenifer/Desktop/DetectionON/imagenet_class_index.json')
labels = {}
print("preds",preds)
#print("preds1",preds1)

# loop over the predictions
for (i, p) in enumerate(preds):
    # grab the prediction information for the current ROI
    (imagenetID, label, prob) = p[0]
    #print("ID",imagenetID)
    #print("label",label)
    #0print("prob",prob)

# loop over the predictions
for (i, p) in enumerate(preds):
    # grab the prediction information for the current ROI
    (imagenetID, label, prob) = p[0]
```

```
        # filter out weak detections by ensuring the predicted probability
        # is greater than the minimum probability
        if prob >= args["min_conf"]:
            # grab the bounding box associated with the prediction and
            # convert the coordinates
            box = locs[i]

            # grab the list of predictions for the label and add the
            # bounding box and probability to the list
            L = labels.get(label, [])
            L.append((box, prob))
            labels[label] = L
    print("label",label)
```

A.2.6. Supresión no máxima

```
# loop over the labels for each of detected objects in the image
for label in labels.keys():
    # clone the original image so that we can draw on it
    print("[INFO] showing results for '{}'".format(label))
    clone = orig.copy()

    # loop over all bounding boxes for the current label
    for (box, prob) in labels[label]:
        # draw the bounding box on the image
        (startX, startY, endX, endY) = box
        cv2.rectangle(clone, (startX, startY), (endX, endY),
            (0, 255, 0), 2)

    # show the results *before* applying non-maxima suppression, then
    # clone the image again so we can display the results *after*
    # applying non-maxima suppression
    cv2.imshow("Before", clone)
    clone = orig.copy()

    # extract the bounding boxes and associated prediction
    # probabilities, then apply non-maxima suppression
    boxes = np.array([p[0] for p in labels[label]])
    proba = np.array([p[1] for p in labels[label]])
    boxes = non_max_suppression(boxes, proba)

    # loop over all bounding boxes that were kept after applying
    # non-maxima suppression
    for (startX, startY, endX, endY) in boxes:
        # draw the bounding box and label on the image
        cv2.rectangle(clone, (startX, startY), (endX, endY),
            (0, 255, 0), 2)
        y = startY - 10 if startY - 10 > 10 else startY + 10
        cv2.putText(clone, label, (startX, y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 2)

    # show the output after apply non-maxima suppression
    cv2.imshow("After", clone)
    cv2.waitKey(0)
```

A.3. Algoritmo de la conexión entre el computador y la cámara de un celular

In [6]:

```
import os, shutil
folder = "C:/Users/Jhenifer/Desktop/DetectionON/ImaGenerator/images"
for the_file in os.listdir(folder):
    file_path = os.path.join(folder, the_file)
    try:
        if os.path.isfile(file_path):
            os.unlink(file_path)
        #elif os.path.isdir(file_path): shutil.rmtree(file_path)
    except Exception as e:
        print(e)


# CREAR DATASET

# get the reference to the webcam
import cv2

# get the reference to the webcam
camera = cv2.VideoCapture(0)
imagesList = []


while(True):

    _, frame = camera.read()

    cv2.imshow("Camera", frame)

    key = cv2.waitKey(1)

    if key & 0xFF == ord("q"):
        break
    elif key & 0xFF == ord("1"):
        imagesList.append(frame)
        print('frame saved')
```

```
camera.release()
cv2.destroyAllWindows()

# GUARDAR IMAGENES
for i, img in enumerate(imagesList):

    import os
    #move to directory containing data
    os.chdir('C:/Users/Jhenifer/Desktop/DetectionON/ImaGenerator')

    image = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

    image = cv2.resize(image, (224, 224))

    cv2.imwrite('./images/{}.jpg'.format(i), cv2.cvtColor(image,cv2.COLOR_BGR2RGB))
```