UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Escuela de Ingeniería en Mecatrónica

Trabajo de Grado Previo a la Obtención del Título de Ingeniero en Mecatrónica

Tema:

"Detección de Enfermedades en Plantas de Tomate a través del Análisis Computacional de sus Hojas"

Autor:**Gabriela Dolores Chiluisa González**

Director:**Carlos Xavier Rosero Chandi**

Ibarra-Ecuador

Marzo 2020

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y APROBACIÓN

A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Arte.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:	100418682-9
APELLIDOS Y NOMBRES:	CHILUISA GONZÁLEZ GABRIELA DOLORES
DIRECCIÓN:	Santa Rosa del Tejar
EMAIL:	gdchiluisag@utn.edu.ec
TELÉFONO FIJO: 2-625-111	TELÉFONO MOVIL: 0998708518
DATOS DE LA OBRA	
TÍTULO:	“Detección de Enfermendades en Plantas de Tomate a través del Analisis Computacional de sus Hojas”
AUTOR:	CHILUISA GONZÁLEZ GABRIELA DOLORES
FECHA (DD-MM-AAAA):	06/07/2021
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	PREGRADO
TÍTULO POR EL QUE OPTA:	INGENIERO EN MECATRÓNICA
ASESOR/DIRECTOR:	CARLOS XAVIER ROSERO CHANDI

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CONSTANCIA

El autor manifiesta que la obra objeto de la presente autorización es original y se 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 06 de julio 2021

Gabriela Dolores Chiluisa González

C.I.: 1004186829

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

En calidad de director del trabajo de grado "DETECCIÓN DE ENFERMEDADES EN PLANTAS DE TOMATE A TRAVÉS DE ANÁLISIS COMPUTACIONAL DE SUS HOJAS" presentado por la egresada GABRIELA DOLORES CHILUISA GONZÁLEZ, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra 06 de julio 2021

Firmado electrónicamente por:
CARLOS XAVIER
ROSERO CHANDI

Carlos Xavier Rosero Chandi

DIRECTOR DE TESIS

Dedicatoria

El presente trabajo de grado lo quiero dedicar, a mi madre, a toda mi familia, a la persona que fructifica mi existencia y a mi misma por haber seguido adelante sin importar el tamaño de los retos que se han presentado.

Agradecimientos

A mis padres por el apoyo incondicional que me brindan cada día, en todos los aspectos de mi vida.

A mi director de trabajo de grado Carlos Xavier Rosero por guiarme a lo largo de este proceso con sus conocimientos y experiencias.

A mis compañeros que hicieron de esta carrera una maravillosa experiencia de vida.

A mi por llegar hasta este gran momento en mi vida, con todas las lecciones aprendidas, siendo este un paso más alcanzado para lograr ser, la persona que me he propuesto ser, profesional, pero sobre todo, un buen ser humano.

Resumen

Los cultivos de todo tipo de plantas están expuestos a enfermedades que pueden darse por: hongos, virus y bacterias, para el caso del tomate no es la excepción, por este motivo se le ha puesto relevancia a contribuir con la detección temprana de enfermedades en las plantas de dichas especies. Se está trabajando en solucionar 4 de la enfermedades que afectan en las plantaciones de tomate. La identificación se puede dar a través de las partes de la planta como son, las hojas, el tallo o los frutos, así que se tomó un data set de imágenes de hojas enfermas. Se extrajo 4 de los 10 tipos con los que se contaba. Se trabaja con el método de redes neuronales convolucionales, dado que son utilizadas para la creación de modelos de clasificación de elementos, entrenados con imágenes. Las imágenes fueron ingresadas a tamaño real de 256x256, pero previamente fotografiadas por una cámara telefónica a fin de obtener el mismo tipo de archivos para entrenar la red y para receptar imágenes en el clasificador. La cámara cuenta con una resolución de 12 Megapíxeles, así que para el entrenamiento, se redimensionaron las imágenes. Al finalizar el entrenamiento se obtuvo un porcentaje de 93 % de eficiencia el cual es admisible y después ingresar los datos de prueba se lograron 429 respuestas correctas y 54 incorrectas, siendo este un porcentaje aceptable. El siguiente paso fue, crear el algoritmo para ejecutar el modelo previamente entrenado y guardado, para luego ser entrenado, ambos procesos son realizados con Python keras y en este último caso, un programa adicional, droid cam para usar la cámara de un motorola g6 plus como web cam, en donde igualmente se obtuvieron buenos resultados en la mayoría de los casos, aunque los dispositivos electrónicos son diseñados para trabajar en un ambiente controlado y al no encontrarse bajo las mismas condiciones no da los mismos resultados. Finalmente es posible ingresar imágenes directamente desde una carpeta de la PC en donde el porcentaje de imágenes aceptadas como correctas son aproximadamente al 93 % que es el valor generado por el entrenamiento y se analizaron los resultados a través de la interacción con el clasificador, ingresando en él imágenes manualmente, para comprobar si el número de elementos clasificados de forma acertada corresponden al porcentaje de eficiencia generado por el entrenador.

Abstract

Crops of all types of plants are exposed to diseases that can occur due to: fungi, viruses and bacteria, in the case of tomato it is no exception, for this reason it has been important to contribute to the early detection of diseases in the plants of these species. Be this working on solving 4 of the diseases that affect tomato plantations. The identification can be given through parts of the plant such as the leaves, the stem or the fruits, so a data set of images of diseased leaves was taken. 4 of the 10 types that were available were extracted. We work with the convolutional neural network method, since they are used to create models for the classification of elements, trained with images. The images were entered at a real size of 256x256, but previously photographed by a telephone camera in order to obtain the same type of files to train the network and to receive images in the classifier. The camera has a resolution of 12 Megapixels, so for training, the images were resized. At the end of the training, a percentage of 93 % of efficiency was obtained, which is admissible and after entering the test data, 429 correct and 54 incorrect answers were achieved, this being an acceptable percentage. The next step was to create the algorithm to execute the previously trained and saved model, for which Python kéras was used as for the trainer algorithm and in the latter case, an additional program, droid cam to use the camera of a motorola g6 plus as a web cam, where good results were also obtained in most cases, although the electronic devices are designed to work in a controlled environment and as they are not under the same conditions, they do not give the same results. It is possible to enter images directly from a folder on the PC where the percentage of images accepted as correct correspond to approximately 93 %, which corresponds to the value generated by the training.

Índice general

Introducción **1**

Problema 1

Objetivos 2

Objetivo general 2

Objetivos Específicos 2

Alcance 2

Justificación 3

1. Revisión literaria **4**

1.1. Identificación de Enfermedades Vegetales a Partir de Lesiones y Manchas Individuales Mediante el Aprendizaje Profundo 4

1.2. Detección de Trastornos en Plantas de Tomate Mediante Aprendizaje Profundo . 5

1.3. Clasificación de Enfermedades de las Plantas de Tomate en Formato Digital Usando Árbol de Clasificación 6

1.4. Comparación de Arquitecturas de Redes Neuronales Convolucionales para la Clasificación de Enfermedades en Tomates 6

1.5. Detector Robusto Basado en Aprendizaje Profundo para el reconocimiento de Plagas y Enfermedades de Plantas de Tomate en Tiempo Real 7

1.6. Redes Neuronales Convolucionales para la Detección y Clasificación de Enfermedades de Plantas en Imágenes Digitales 8

1.7. Clasificación de Clorosis en Hojas de Árboles de Naranja Mediante Aprendizaje Automático 8

1.8. Propuesta 8

2. Metodología **10**

2.1. Elección del set de datos . . . 11
2.2. Ingreso de los Datos . . . 12
2.3. Procesamiento de Matrices . . . 13
2.4. Pre-procesamiento de Imágenes . . . 14
2.5. Entrenamiento CNN . . . 14
2.5.1. Funciones de Activación . . . 15
2.5.2. Parámetros para el Entrenamiento . . . 17
2.6. Prueba de eficiencia del modelo entrenado . . . 17
2.7. Clasificador . . . 19
2.7.1. Ingreso Manual . . . 19
2.7.2. Recepción de imágenes por medio de cámara web . . . 19
2.8. Equipo y Software . . . 20

3. Resultados **22**
3.1. Entrenamiento . . . 22
3.2. Aplicación . . . 23
3.2.1. Resultados y Pruebas . . . 23

4. Conclusiones y Recomendaciones **30**
4.1. Conclusiones . . . 30
4.2. Recomendaciones . . . 31
4.3. Trabajo Futuro . . . 31
4.4. Código del entrenador CNN . . . 35
4.5. Código del Clasificador . . . 40

Introducción

Problema

En el Ecuador, la agricultura es uno de los ejes principales de la economía [1], a finales del 2018 representó un 8 por-ciento del Producto Interno Bruto (PIB) [2] y durante los últimos años se ha trabajado por incrementar la productividad y así cumplir con la demanda interna y externa. Uno de los productos con mayor demanda nacional es el tomate riñón, en el 2017 Loja, Guayas e Imbabura fueron las provincias con la mayor producción de este tipo de tomate [12] y solo en esta última se registró una producción anual del 30.43 por-ciento, con 19 073 toneladas [3], cifras que no permitieron registrar importaciones en ese año [12].

Cada ecuatoriano consume, en promedio, 4 kilos de tomate riñón al año, ya sea crudo en ensaladas, cocinado para darle sabor a las comidas o industrializado en forma de salsa. El tomate es muy apetecido por ser un alimento de fácil digestión y rico en vitaminas A, B y C, fósforo, potasio, hierro, calcio y licopeno [5]. A esto se puede agregar que la producción de tomate riñón en el cantón Ibarra es probablemente la más baja comparada con la de otros cantones productores, pero contrariamente a la parte productiva, es el más importante desde el punto de vista comercial ya que en el mercado mayorista de la ciudad de Ibarra es donde se comercializa la mayor parte del producto que se cultiva en la provincia [4].

Sin embargo, es importante mencionar que la producción de tomate está siendo afectada por una gran cantidad de bacterias, virus, hongos y plagas diferentes [6]. Una sola planta es capaz de infectar a cultivos enteros haciendo que se pierdan cifras significativas del cultivo, ocasionando pérdidas de orden económico a los agricultores [9]. El costo para controlar plagas y enfermedades puede llegar a alcanzar hasta el 20 por-ciento del valor de la producción [7]. Además, el manejo de plagas y enfermedades con agro-químicos puede resultar perjudicial para

la salud de los agricultores y de los consumidores [8]

Hoy en día, se puede afirmar que los avances tecnológicos ocurridos en los últimos años han supuesto una nueva era para la agricultura, la cual ha sabido adaptarse a los nuevos cambios, introduciendo a la metodología tradicional, la tecnología informática que existe, dando como resultado nuevas técnicas de cultivo como son la agricultura de precisión y los cultivos hidropónicos [10]. Si a esto se añade tecnología de visión artificial, se puede obtener un control total sobre los cultivos por medio de imágenes para su posterior análisis e interpretación [11] siendo esta una buena opción para el reconocimiento de defectos en plantas de tomate lo cual contribuye a la reducción de su propagación y a la vez permite la erradicación temprana sin perder grandes escalas de producción

Desarrollar un algoritmo de identificación de varios tipos de anomalías en hojas de plantas de tomate por medio de visión artificial.

- Analizar el contexto del problema en base a las imágenes seleccionadas para escoger la técnica de aprendizaje de máquina más adecuada y eficiente.
- Desarrollar un programa de clasificación de defectos por medio del análisis de imágenes.
- Validar el programa de reconocimiento de defectos a través de imágenes de la base de datos que no sean utilizada para el aprendizaje de máquina.

El presente proyecto propone el desarrollo de un algoritmo, para lo cual se seleccionará un método con el que se realizará el análisis computacional de las imágenes, para reconocer en ellas características peculiares. Realizar el aprendizaje de máquina a través del método de clasificación, para detectar anomalías similares y clasificarlas acorde a la enfermedad de tomate a la que pertenezcan, empleando para este fin una base de datos ya existente. Se validará el

correcto funcionamiento del algoritmo haciendo uso de una parte de la base de datos, no tomada en cuanta para el aprendizaje de máquina.

Se realizará un algoritmo de identificación de anomalías en hojas de tomate, debido al problema encontrado en cuanto a que las plantas de tomate son susceptibles a diversas enfermedades, causadas por distintos factores usualmente ambientales, esto a través del uso de los conocimientos adquiridos durante la formación en la carrera de Ingeniería en Mecatrónica.

Se realizará el filtrado de las imágenes de la base de datos obtenida desde repositorios, para luego llevarlas al proceso de aprendizaje de máquina, dando como resultado la identificación de defectos. La utilidad del presente proyecto se destaca en que beneficiará a los productores de tomate, para la detección temprana de enfermedades que afectan a las plantas de sus cultivos con el fin de evitar el deterioro de las mismas, ya que podrán ser tratadas a tiempo, contribuyendo a que se generen menos pérdidas en la cosecha y comercialización del producto.

Además, de que ayudará al desarrollo de futuros trabajos relacionados con agricultura de precisión que se enfoquen con visión artificial y aprendizaje de máquina, como por ejemplo detección de defectos en frutos que también es una problemática frecuente, entre otras.

Capítulo 1

Revisión literaria

Dentro del campo de inteligencia artificial y las múltiples partes que lo conforman, está el aprendizaje automático y la visión artificial, para los cuales existen una variedad de métodos a elegir dependiendo de la aplicación que se le desee dar.
En este caso se planea realizar un algoritmo de clasificación por medio de imágenes para el reconocimiento de enfermedades en plantas de tomate, para lo cual se han estudiado trabajos realizados anteriormente, en donde se reconoce por su alto porcentaje de precisión, al uso de redes neuronales en distintos programadores, en donde dependiendo de los parámetros dados al sistema utilizado, el porcentaje de exactitud varía.

1.1. Identificación de Enfermedades Vegetales a Partir de Lesiones y Manchas Individuales Mediante el Aprendizaje Profundo

En este trabajo se estudió el reconocimiento de 79 enfermedades de 14 especies de plantas (incluido un cultivo de tomate), en donde por la falta de elementos en la base de datos se utilizó la técnica de aumento de datos. Este estudio se destaca por el uso de únicamente las partes lesionadas y con manchas, en lugar de considerar toda la hoja, con lo que se logra que el número de datos no sean un problema, debido a que cada región analizada cuenta con características distintas. La segmentación de los datos sigue siento realizada de forma manual, impidiendo que el sistema sea totalmente automático. Se puede destacar que la eficiencia al clasificar no es muy buena, ya que se obtuvo en cada cultivo una precisión superior al 75 %. El proceso por el cual pasaron las imágenes se puede visualizar en las siguientes figura 2.1 [13].

Figura 1.1: a)Ejemplo de síntomas grandes dispersos b) lesión espúrea desaparecida y c) lesión espúrea sin cambios (c)[13].

1.2. Detección de Trastornos en Plantas de Tomate Mediante Aprendizaje Profundo

Este proyecto parte de la unión entre aprendizaje profundo y redes neuronales, obteniendo: redes neuronales profundas; para la detección de enfermedades se ha recopilado imágenes descargadas de Internet, datos estándar de una aldea vegetal y fotos de un ambiente natural no controlado, dando un total de 13,548 imágenes (de 256x256 pixeles) para el reconocimiento de dos enfermedades (tizón tardío, tizón temprano) y hojas sanas, con lo que se tiene tres clases, en la figura 2.2, se puede observar el conjunto de muestra de los datos. Al tener un alto número de imágenes para el reconocimiento, la precisión de este trabajo llega al 97.25 %, siendo este un sistema validado con buenos resultados [14].

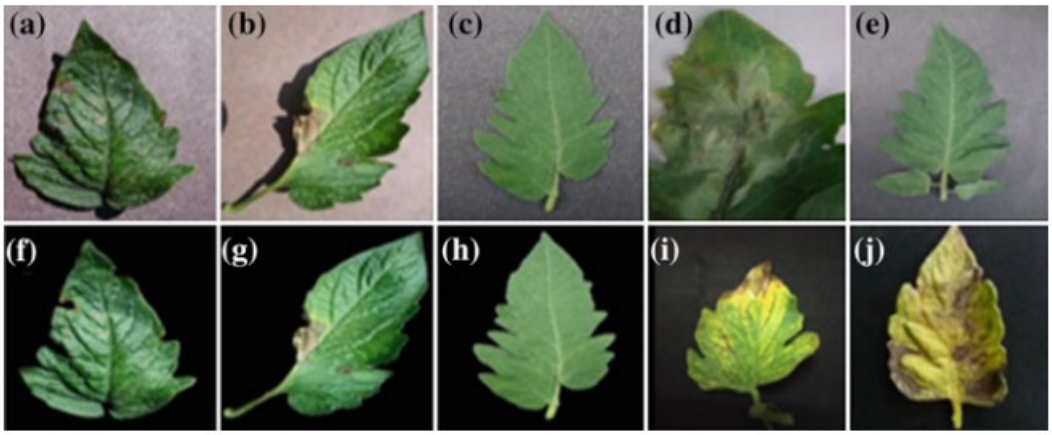

Figura 1.2: Imágenes de muestra del conjunto de datos de la aldea de plantas, el conjunto de datos de Internet y el conjunto de datos del mundo real que se utilizan en varias configuraciones experimentales (a-e), imágenes no segmentadas (F-j) e imágenes segmentadas [14].

1.3. Clasificación de Enfermedades de las Plantas de Tomate en Formato Digital Usando Árbol de Clasificación

En este caso se ha utilizado un método diferente para el reconocimiento de enfermedades, que es Árbol de Clasificación, con el cual se reconocen 5 enfermedades de las plantas de tomate como son: tizón tardío del tomate, mancha de septoria, mancha bacteriana, cancro bacteriano, rizo de hoja de tomate; haciendo uso de imágenes digitales, tanto de hojas como de los tallos de la planta para la identificación de características (color, forma, textura), todo esto tras un proceso de segmentación por medio del método de Otsu, aplicado a una base de datos de 383 imágenes, sin tomar en cuenta las sección de pruebas para lo que se apartaron 39 imágenes. En la siguiente figura 2.3, se puede apreciar de mejor manera el proceso llevado a cabo [15].

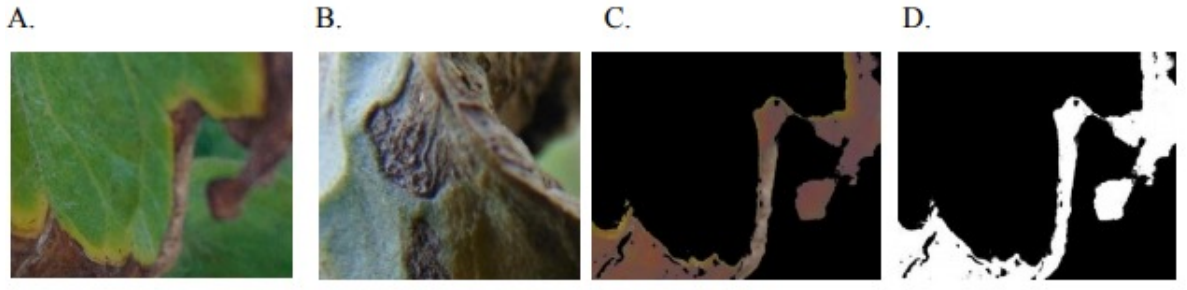

Figura 1.3: A) y B) Hojas de tomate afectadas por el tizón tardío C) segmentación color D) segmentación binaria [15].

1.4. Comparación de Arquitecturas de Redes Neuronales Convolucionales para la Clasificación de Enfermedades en Tomates

Para este trabajo se entrenaron distintas arquitecturas con redes neuronales convolucionales: AlexNet, GoogleNet, Iception V3, ResNet 18 y ResNet 50. Haciendo uso de una base de datos de Plant Villaje que cuenta con 54,323 datos de 14 cultivos y 38 enfermedades de donde se extrajo 18,160 imágenes pertenecientes a cultivos de tomate que cuenta con 10 clases: tomate sano, mancha bacteriana, tizón temprano, tizón tardío, mancha negra, mancha de la hoja,araña roja, mancha blanca, virus del mosaico, rizado amarillo del tomate, que se pueden visualizar en la figura 2.4.

Cabe mencionar, que en los modelos entrenados se utilizó descenso de gradiente estocástica.

Además de transferencia de aprendizaje con el fin de mejorar la eficiencia obteniendo una precisión mayor a 98 % en todos los casos, que se evaluó por matriz de confusión multiclase. La secuencia utilizada y algunos ejemplos de la base de datos se visualiza a continuación.[16].

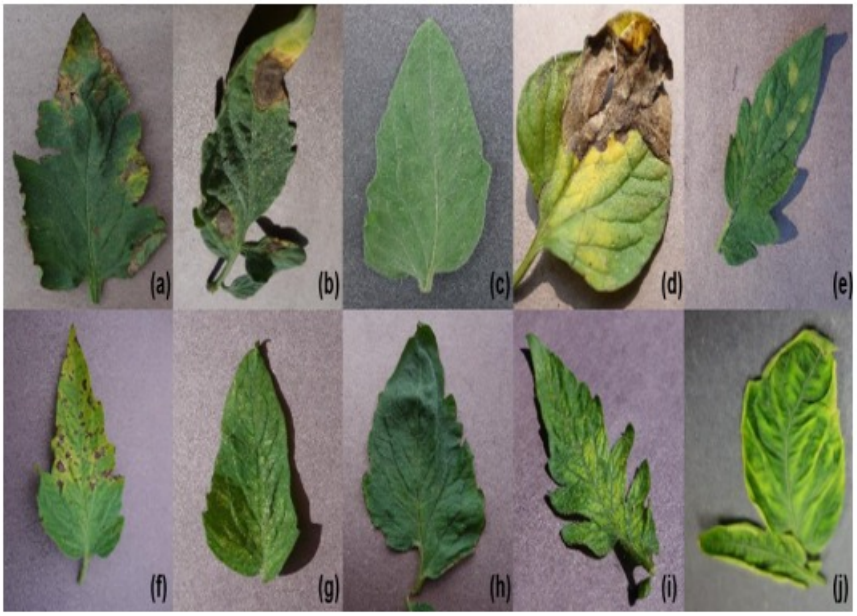

Figura 1.4: Imagenes de muestra obtenidas de PlantVillage - Dataset [16].

1.5. Detector Robusto Basado en Aprendizaje Profundo para el reconocimiento de Plagas y Enfermedades de Plantas de Tomate en Tiempo Real

Las imágenes adquiridas para es trabajo se capturaron en un ambiente no controlado, con cámaras de varias resoluciones, con las que se captaron al rededor de 5000 fotografías de hojas enfermas de: moho gris, cancro bacteriano, moho de la hoja, plaga, minador de hojas, mosca blanca, baja temperatura y moho polvoriento; las cuales se dividió en 80 % para entrenamiento, 10 % para validación y 10 % para pruebas. Se hizo uso de redes neuronales convolucionales de tres tipos: Faster R-CNN, R-FCN, SSD; mismas que fueron combinadas con varios extractores de características profundas como son: VGG net, ResNet, contribuyendo a disminuir la cantidad de falsos positivos durante el entrenamiento y realizando un aumento en la base de datos para evitar el sobre ajuste.[17]

1.6. Redes Neuronales Convolucionales para la Detección y Clasificación de Enfermedades de Plantas en Imágenes Digitales

En este caso se ha realizado el reconocimiento de 24 enfermedades de 9 especies de cultivos, con técnicas de visión artificial. La parte de la planta utilizada son las hojas de las cuales se obtuvo imágenes a color para el reconocimiento de patrones, a través de varios clasificadores para comparar su efectividad como son: discriminante de análisis lineal, máquina de soporte vectorial, y redes neuronales probabilísticas, que se encuentran implementadas en las siguientes arquitecturas:AlexNet, GoogleNet, Inception V3, SqueezeNet, ResNet 101, ResNet 50; de donde se identificó a AlexNet como la de mayor exactitud, debido a que alcanzó el 98.90 % de efectividad[18].

1.7. Clasificación de Clorosis en Hojas de Árboles de Naranja Mediante Aprendizaje Automático

Se han tomado 60 fotografías de hojas saludables y 60 fotografías con clorosis sintomática, tomadas con una cámara de 5 Megapíxeles en un ambiente controlado para reducir brillo y reflectancia. Las características se han obtenido mediante algoritmos de reconocimiento de patrones, aplicando procesamiento digital de imágenes, en conjunto con el método de OTSU para la segmentación, para eliminar el fondo de la imagen, seguido de una conversión a escala de grises. Los nuevos datos son guardados en formato CSV, en donde cada columna contiene los atributos de cada hoja y las 6 últimas pertenecen a las características estadísticas, datos que son ingresados en los meta clasificadores de los lenguajes Matlab y Weka V3.8 combinados con el algoritmo Dl4jMlp, estos datos dieron como resultado una precisión del 100 % en todos los casos [19].

1.8. Propuesta

Se propone el desarrollo de un clasificador de enfermedades en plantas de tomate para lo que se hará uso de imágenes de hojas sanas y de varias clases de hojas enfermas, aplicando machine learning para la creación de un algoritmo al cual se entrenará, para su posterior uso

en la lectura y detección de enfermedades mediante visión artificial.

Capítulo 2

Metodología

En este capítulo se detalla el funcionamiento de las redes neuronales convolucionales, como también las partes que conforman una red CNN, lo que se puede apreciar en el siguiente diagrama de flujo de la figura 2.1.

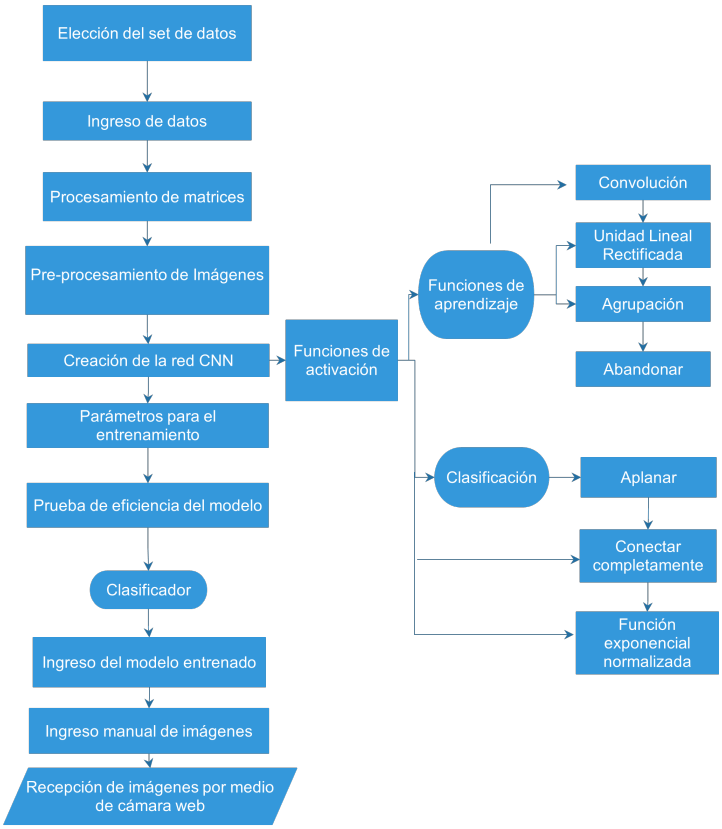

Figura 2.1: Diagrama de flujo del procedimiento para la creación del presente proyecto

2.1. Elección del set de datos

A través de las páginas de internet como: kaggle, Reddit, github, entre otras se realiza la elección del set de datos acorde a la necesidad, como el caso de las enfermedades de plantas de tomate, para lo que es necesario una muestra que contenga imágenes, en donde se pueda reconocer que una planta o una parte de esta, contiene signos de ciertos tipos de enfermedad.

Tras la búsqueda de los datos que se requiere para el entrenamiento, se ha elegido una carpeta con 10 separaciones que contienen imágenes de enfermedades del tomate; la parte de la planta en donde se puede identificar las anomalías son: tallos, hojas, frutos; en este caso el set de datos contiene fotografías de hojas de la planta, las mismas que se diferencian de acuerdo con el tipo de mancha. Dentro de un set de datos un aspecto importante es el cantidad de elementos con los que se trabaja y para el caso de las redes neuronales convolucionales, entre más imágenes ingresan la eficiencia aumenta, logrando que la red identifique de mejor manera al presentar muestras con las que no haya tenido interacción con anterioridad. Para esta ocasión se hizo uso de 4 (bacterial spot, earli blight, healthy y septoria leaf spot) de las 10 enfermedades mencionadas, tomando alrededor de 600 elementos por carpeta, de un total de 13,676 datos. Debido a que, con una mayor cantidad de enfermedades, también es necesario un mayor costo computacional y tomando en cuenta las características de la máquina con la que se trabajó, esto contribuyo a dejar una gran holgura en los datos correspondientes a la parte de comprobación de resultados. las carpetas utilizadas para el entrenamiento se pueden visualizar en la figura 2.2.

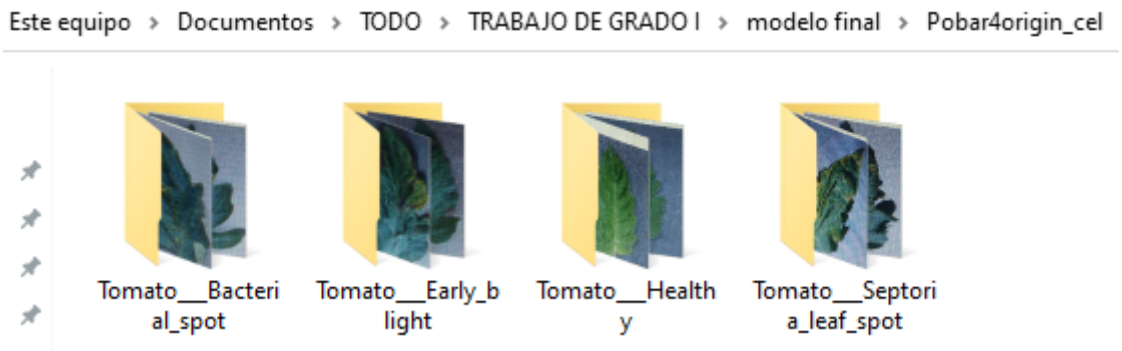

Figura 2.2: Archivos del data set para el entrenamiento (tres clases de tomates enfermos y una clase de tomate sano)

2.2. Ingreso de los Datos

Las imágenes se encuentran en un formato JPG, con un tamaño predeterminado de 256x256, mismas que luego tendrán que ser identificadas por el algoritmo clasificador, por esta razón las imágenes se fotografiaron con la cámara telefónica, que posteriormente sería la receptora de las imágenes para el algoritmo clasificador en un ambiente controlado con el fin de que ingrese la menor cantidad de ruido a las imágenes. Esto se puede visualizar en la figura 2.3 en donde se encuentra la imagen tal como se ha descargado de la página web kaggle y en la figura 2.4, la fotografía de la misma. Como se puede notar la relación de tamaño en las imágenes ha cambiado, ahora la imagen fotografiada tiene una dimensión de 3024 de ancho por 4032 de alto, siendo 4 veces más grande que la original, dado esto se realizó un re-dimensionamiento de las imágenes tras su ingreso en el programa para tenerlas nuevamente en un tamaño de 256x256, y así evitando un mayor costo computacional.

La red neuronal busca patrones y en cada convolución se adentra en modelos más complejos por esta razón es importante que la imagen no pierda su forma; mientras los contornos se puedan reconocen y diferencias unos de otros, la red realizará de forma optima su trabajo.

Las imágenes deben estar bajo las mismas condiciones tanto en el ambiente que se han creado, como también a la hora de ser procesadas previo al entrenamiento y a la clasificación, debido a que el durante el entrenamiento el programa aprendió a reconocer imágenes de un cierto tipo y al ser utilizado en el programa de clasificación buscará la relación entre lo que aprendió y lo que se le presenta; y si el cambio entre estas las dos entradas al sistema es demasiado incompatible no se obtendrá el resultado esperado, entonces, mientras las entradas dadas al clasificador sean equivalentes en fondo y forma a las dadas al entrenamiento, la eficiencia al clasificar será la esperada.

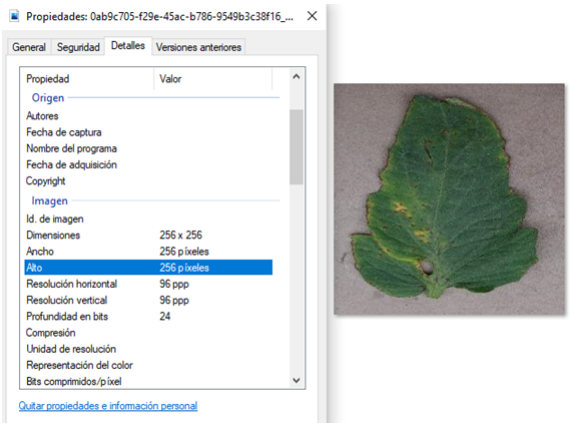

Figura 2.3: Propiedades de las imágenes originales del data set.

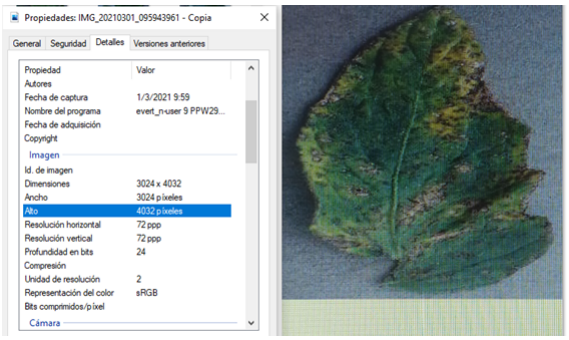

Figura 2.4: Propiedades de las imágenes del nuevo data set, captadas por la cámara telefónica.

2.3. Procesamiento de Matrices

Primero se ha dado a cada enfermedad un número, correspondiente a la clase que pertenece esto se muestra en la 2.5.

```
0 Tomato___Bacterial_spot
1 Tomato___Early_blight
2 Tomato___Healthy
3 Tomato___Septoria_leaf_spot
```

Figura 2.5: Asignación de clases a cada enfermedad.

Para introducir las imágenes al modelo de entrenamiento se crea a partir de ellas una matriz que corresponde a las etiquetas de cada clase, en este caso fueron 4, una por cada enfermedad y en conjunto se crea otra en donde se encuentran las matrices formadas por cada imagen que ha ingresado. A dichas matrices se le ha realizado una división para generar tres secciones, que corresponden a las partes de entrenamiento, validación y prueba del algoritmo, se ha considerado

que la división se realice en porcentajes de 80 % para entrenamiento y validación, dejando el 20 % restante para la fase de prueba 2.6 [21].

```
Training data shape :  (1932, 256, 256, 3) (1932,)
Testing data shape :  (483, 256, 256, 3) (483,)
```

Figura 2.6: División de la muestra en 80 % (1932 imágenes) para entrenamiento y 20 % (483 imágenes) para prueba.

2.4. Pre-procesamiento de Imágenes

En redes neuronales se entiende que, a mayor cantidad de datos, mejor clasificación de clases, dicho esto las imágenes pasan por una etapa en la que se genera lo que se conoce como aumento de data set, para esto a cada imagen del conjunto de entrenamiento y validación, se las introduce en un método ImagenDataGenerator() en donde se les realiza un zoom de un 20 %, una rotación 10 grados, un desplazamiento horizontal de 0.1, un desplazamiento vertical de 0.1 y un rango de corte de 0.1, con lo que se obtienen 5 imágenes adicionales por cada una de las ingresadas.

2.5. Entrenamiento CNN

Las redes neuronales convolucionales trabajan con modelos de aprendizaje supervisados, en donde se presenta como el agente que se supervisa a quien da la entrada al sistema a partir de la cual debe darse una salida predeterminada [22].

Por norma general, el proceso de aprendizaje parte de un conjunto de pesos sinápticos aleatorio y busca un conjunto de pesos que permitan a la red desarrollar correctamente una determinada tarea. Durante el proceso se va refinando iterativamente la solución hasta alcanzar un nivel de operación suficientemente aceptable [11]. Las redes neuronales convolucionales están creadas esencialmente por una serie de capas que buscan identificar bordes, para luego adentrarse en combinación de bordes y finalmente modelos de objetos. Receptan grandes cantidades de imágenes de las cuales se extraen características exclusivas de cierto tipo de elemento con el fin de generalizarlo. Al ingreso de la imagen y conociendo sus dimensiones, se forma una matriz que contiene ancho, alto y profundidad, la que es llenada con datos numéricos que van de 0 a 255, pero se cambian a valores entre 0 y 1 debido a que así es como los acepta la red creada en Python, para la ejecución del entrenamiento [24].

2.5.1. Funciones de Activación

Convolución

La convolución lo que hace es generar un volumen de imágenes, es decir aumenta la profundidad según la cantidad de filtros que se apliquen, no debería reducir la dimensión espacial. Ahora para obtener las características de las imágenes la convolución trabaja con un kernel que es el que se traslada por toda la imagen, realizando multiplicación matricial, seguido de una media ponderada con la matriz de las dimensiones que se le asignan al parche, que para este caso se ha escogido un tamaño de 3x3. Tras pasar el kernel por las imágenes, estas cambian de dimensión debido a las operaciones que se realizan en ellas, quedando disminuidas en varias filas y columnas dependiendo del tamaño de la zancada que se realice en cada salto del filtro. Para solucionar el conflicto de la reducción de la dimensión espacial, se hace uso de función padding, para rellenar los bordes de las imágenes con ceros tantas filas y columnas sea necesario para obtener la dimensión inicial de la imagen que es de 256x256.

La convolución es una operación matemática que trabaja con dos funciones (f y g) y produce una tercera (h), que es una integral que expresa la cantidad de superposición de una función a medida que se desplaza sobre la otra función. Denotada por la Ecuación 2.1 [16].

$$h = f * g = \int_{\infty}^{\infty} f(\tau) * g(t - \tau) * dy \tag{2.1}$$

Agrupación

En keras esta función tiene el nombre de MaxPolling, la misma que no afecta a la profundidad de la imagen, sin embargo, si a las dimensiones espaciales de ancho y largo, a consecuencia de que se crea un nuevo filtro que es de 2x2 con zancada de 2, mismo que recorre la imagen de izquierda a derecha, iniciado desde la parte superior cada muestra y realizando una operación que promedia las cifras, entregando solo un número, el más alto para la nueva matriz, la cual será de 128x128, misma que ha de ser la capa oculta (llamada así debido a que no se tiene acceso directo a su salida, que serían todas a excepción de la última) de entrada para la siguiente función. El trabajo que se realiza en las matrices que puede observar en la figura 2.7.
El número de neuronas se incrementa de forma exponencial según el número de las capas ocultas va aumentando y profundizando, con ello el costo computacional también aumenta, siempre manteniendo las características detectadas en las anteriores capas.

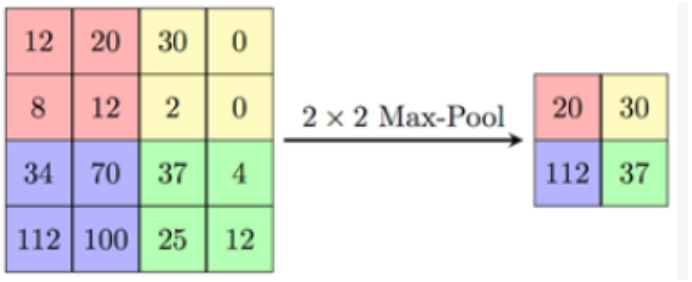

Figura 2.7: pooling.

Unidad lineal rectificada

Se hace uso de la capa leakyRelu, entendiendo primero que en las matrices de las imágenes pueden existir valores tanto positivos como negativos, la presente función se encarga de multiplicar únicamente a los valores negativos por un coeficiente de rectificado para tener valores positivos en toda la matriz.

Abandonar

Es la capa encargada de contribuir a que el sobre ajuste en la red neuronal sea el menor posible debido a que desactiva una cantidad determinada de neuronas que pertenecen a la red, en este caso se desactivan la mitad de las neuronas utilizadas en cada interacción de forma aleatoria. Esta función de activación solo se realiza para el entrenamiento; la validación y pruebas cuenta con todas las neuronas activas.

Aplanar

Al conseguir el tamaño más pequeño de matriz que es el resultado de las operaciones antes mencionadas, se procede a aplanar la matriz, convirtiéndola en un vector.

Función exponencial normalizada

Existen múltiples variaciones que se puede realizar, dependiendo de la dimensión de las matrices en la CNN se efectúa lo que es generalmente recomendable, tratar de reducir la dimensionalidad, pasando de 3D a 2D y en lo posible a una sola dimensión, esto se utiliza en la salida de las etiquetas de clase en donde a la salida se recepta datos en formato one hot. Aquí ingresan todos los datos de probabilidad que ha designado el entrenador y estos se traducen a valores entre 0 y 1, para luego escoger la probabilidad más alta de que un dato pertenece a una clase específica. Esta se conecta directamente con la última capa, la de clasificación.

Clasificación

Esta capa es la última, en donde dependiendo de qué tan alta sea la probabilidad del objeto de pertenecer a una clase, se tomará el valor más alto y se le asignará esa probabilidad. Internamente los pesos y el error se actualizan mediante el algoritmo de backpropagation.

2.5.2. Parámetros para el Entrenamiento

Inicializar el entrenamiento es el paso final al crear la red, para esto se deben escoger los parámetros con los que se desea ejecutarlo, como son: La categoría multiclase, que la ejecución de esta función incurre en que la salida del sistema debe pertenecer únicamente a una de las clases existentes. En cada ocasión que el sistema ha sido correctamente entrenado, al finalizar el aprendizaje se desconecta. Esto quiere decir, que los pesos y la estructura de la red quedan fijos, quedando la red neuronal dispuesta para el procesamiento de datos [23].

2.6. Prueba de eficiencia del modelo entrenado

Para evaluar el desempeño de la red, sklearn destina un porcentaje del conjunto de datos nombrado como "prueba" que para este caso se ha designado un 20 % de la muestra total, siendo 483 imágenes que son ingresadas automáticamente al modelo una vez que se ha entrenado, con lo que se obtiene dos listas de salida, una con el total de clasificaciones identificadas correctamente figura 2.8. y otra lista con las muestras clasificadas de forma incorrecta figura 2.9, que se guardan en dos archivos scv, con dos columnas, la primera en donde se encuentran las etiquetas reales de cada imagen y otra con las etiquetas asignadas por el modelo entrenado. La evaluación de desempeño puede ser tanto automática, como manual, mismas que se han realizado por los dos métodos, para la verificación manual se han usado los datos de la parte de comprobación obteniendo muy buenos resultados.

	Filename	Predictions
2	Tomato___Septoria_leaf_spot	Tomato___Septoria_leaf_spot
3	Tomato___Septoria_leaf_spot	Tomato___Septoria_leaf_spot
4	Tomato___Healthy	Tomato___Healthy
5	Tomato___Septoria_leaf_spot	Tomato___Septoria_leaf_spot
6	Tomato___Septoria_leaf_spot	Tomato___Septoria_leaf_spot
7	Tomato___Healthy	Tomato___Healthy
8	Tomato___Septoria_leaf_spot	Tomato___Septoria_leaf_spot
9	Tomato___Bacterial_spot	Tomato___Bacterial_spot
10	Tomato___Septoria_leaf_spot	Tomato___Septoria_leaf_spot
11	Tomato___Healthy	Tomato___Healthy
12	Tomato___Early_blight	Tomato___Early_blight
13	Tomato___Early_blight	Tomato___Early_blight
14	Tomato___Bacterial_spot	Tomato___Bacterial_spot
15	Tomato___Septoria_leaf_spot	Tomato___Septoria_leaf_spot
16	Tomato___Bacterial_spot	Tomato___Bacterial_spot
17	Tomato___Healthy	Tomato___Healthy
18	Tomato___Healthy	Tomato___Healthy
19	Tomato___Bacterial_spot	Tomato___Bacterial_spot
20	Tomato___Bacterial_spot	Tomato___Bacterial_spot
21	Tomato___Healthy	Tomato___Healthy

correct

Figura 2.8: Datos correctos adquiridos de ingresar la parte de prueba al entrenamiento.

	Filename	Predictions
2	Tomato___Early_blight	Tomato___Bacterial_spot
3	Tomato___Early_blight	Tomato___Bacterial_spot
4	Tomato___Healthy	Tomato___Early_blight
5	Tomato___Early_blight	Tomato___Bacterial_spot
6	Tomato___Healthy	Tomato___Early_blight
7	Tomato___Septoria_leaf_spot	Tomato___Bacterial_spot
8	Tomato___Early_blight	Tomato___Bacterial_spot
9	Tomato___Early_blight	Tomato___Septoria_leaf_spot
10	Tomato___Early_blight	Tomato___Bacterial_spot
11	Tomato___Bacterial_spot	Tomato___Early_blight
12	Tomato___Septoria_leaf_spot	Tomato___Early_blight
13	Tomato___Early_blight	Tomato___Bacterial_spot
14	Tomato___Septoria_leaf_spot	Tomato___Bacterial_spot
15	Tomato___Early_blight	Tomato___Bacterial_spot
16	Tomato___Healthy	Tomato___Early_blight
17	Tomato___Septoria_leaf_spot	Tomato___Bacterial_spot
18	Tomato___Early_blight	Tomato___Bacterial_spot
19	Tomato___Healthy	Tomato___Early_blight
20	Tomato___Early_blight	Tomato___Septoria_leaf_spot
21	Tomato___Early_blight	Tomato___Bacterial_spot

incorrect

Figura 2.9: Datos incorrectos adquiridos de ingresar la parte de prueba al entrenamiento.

2.7. Clasificador

El clasificador está conformado por dos partes, una en la que se ingresan imágenes directamente de una carpeta creada en el ordenador, llamada mediante una extensión del lugar en donde se encuentra la imagen, la misma que es procesada y como resultado se entrega una matriz con las probabilidades de que pertenezca a cada clase. Además, de imprimir el nombre de la imagen ingresada, junto al nombre de la etiqueta con la probabilidad de la clase más alta. Tanto para el ingreso manual como para la recepción de imágenes por medio de cámara web es necesario que el modelo se encuentre previamente en el código para lo que se insertan todas las librerías aplicadas para el algoritmo entrenador a fin de luego llamar al archivo y los pesos del modelo que fue guardado.

2.7.1. Ingreso Manual

Las imágenes que se han designado para la comprobación final de eficiencia del modelo son ingresadas desde una carpeta llamada "Comprobación" y cada una tiene un nombre, el cual debe actualizarse cada que se carga una imagen distinta, misma que se redimensiona a 256x256, que es el mismo tamaño con el que se entrenó la red, de igual forma se le hace todas las conversiones realizadas a las imágenes previas a entrar a la red. Tras pasar por el modelo entrenado, emite una lista en donde se muestra el porcentaje probabilístico de cada clase. Además, del nombre de la imagen ingresada junto a la designación de la clase con el porcentaje más alto de la lista antes mencionada.

2.7.2. Recepción de imágenes por medio de cámara web

Primero deben configurarse los parámetros de la cámara como: largo, ancho y brillo de la imagen que se receptará. Para conseguir imágenes con mejor resolución en lugar de hacer uso de la cámara web de la computadora, se hizo uso de la cámara de un teléfono celular (motorola G6 plus) para lo que se usa el código IP del celular con lo que se accede a la cámara desde el programa, una vez conseguido eso se configura los parámetros gráficos de la pantalla de recepción en donde aparecerán directamente el nombre de la clase a la que corresponde la imagen de la hoja que se muestra a la cámara, junto a la etiqueta que le corresponde [24].

2.8. Equipo y Software

Para iniciar se destaca que el uso de anaconda se implementa debido a que permite el desarrollo de proyectos relacionados con ciencias de datos haciendo uso de diversos entornos, para este caso se utiliza Jupyter [9], dentro de esta multiplataforma de forma interna, se hace uso de los recursos del sistema que vienen a ser el CPU, que se encarga de tareas de todo tipo que le son asignadas como cálculos y la GPU, misma que se encarga netamente del procesamiento de imágenes que contribuye a que el procesador se libere de estas tareas, mejorando el rendimiento y reduciendo el tiempo en que se desempeñas los procesos, obteniendo un entrenamiento logrado en un tiempo más corto y evitando un costo computacional más elevado. A continuación en la 2.10, se puede visualizar las características con las que cuenta la máquina que sé ha utilizado en este proyecto [24]

También se ha hecho uso de un teléfono celular como cámara web, para obtener imágenes con buena resolución que el clasificador pueda reconocer de una forma más sencilla y sus características se muestran a continuación en la 2.11, se toma en cuenta principalmente la resolución de la cámara trasera la cual cuenta con una resolución de 12 Megapíxeles, que se puede comparar también con las características de la cámara que es parte de la unidad laptop en donde se puede reconocer que se tiene 0.307 Megapíxeles como se muestra en la figura 2.12.

MARCA	ASUS
REFERENCIA	K401UQ-GA159T
PROCESADOR	Intel® Core□ i5-6200U Processor, 2.3GHz
GENERACIÓN PROCESADOR INTEL	Intel 6ta Generación
SISTEMA OPERATIVO	Windows
MEMORIA RAM	8 GB
PULGADAS	14 Pulgadas
TARJETA DE VIDEO	NVIDIA GT940MX 2GB DDR3
UNIDAD ÓPTICA	No tiene
CONECTIVIDAD	BLUETOOTH, HDMI, USB, WIFI

Figura 2.10: Caracteristicas del equipo laptop utilizado.

CÁMARA	Principal	Dual, 12MP (f/1.7) + 5MP (f/2.2)
	Funciones	autofocus por detección de fase Pixel Dual, flash LED dual, foco táctil, detección de rostro y sonrisa, geo-tagging, HDR
	Video	2160p@30fps, 1080p@30/60fps
	Frontal	8 MP, flash LED, 1080p

Figura 2.11: Características del teléfono utilizado, cámara trasera.

Información de la cámara web	
Nombre de la webcam:	USB2.0 VGA UVC WebCam
Calificación de calidad:	14
Micrófono incorporado:	None
Altavoz incorporado:	None
Cuadros por segundo:	5 FPS
Tipo de corriente:	video
Modo de imagen:	rgb
Webcam Megapíxels:	0.31 MP
Resolución de cámara web:	640×480
Video estándar:	VGA
Relación de aspecto:	1.33
Tamaño de archivo PNG:	350.88 kB
Tamaño de archivo JPEG:	214.71 kB
Velocidad de bits:	1.05 MB/s

Figura 2.12: Características cámara web del dispositivo asus.

Para que sea posible el uso de la cámara telefónica, como cámara web, se ha utilizado un software, llamado Droid cam, con el cual se hace la conexión desde la cámara al celular, cabe mencionar que en ambos dispositivos debe constar la instalación del mencionado software y la conexión puede ser con cable USB o por una red wifi (los dos dispositivos, deben estar conectados a la misma red wifi), éste es un software libre que se encuentra la app store de Android.

Capítulo 3

Resultados

3.1. Entrenamiento

Continuación se describen los procesos ejecutados; la forma más adecuada de cargar las imágenes al programa se realizó mediante prueba y error.

Primero se realizó la división manual de datos para entrenamiento, validación el 80 % y para prueba el 20 %, con imágenes a la que se cambió su tamaño de 256x256 pixeles a 300x300 a las cuales se les realizó ciertas operaciones como zoom, rotación, espejo con el fin de obtener más datos y que el entrenamiento identifique de mejor manera las enfermedades en las hojas de tomate. Usando una red neuronal convolucional formada por dos capas ocultas y 25 épocas, esto no fue suficiente para obtener resultados óptimos, ya que la eficiencia alcanzada apenas llegaba al 60 % en teoría, pero a la hora de revisar los datos de prueba existía una inconsistencia total entre las imágenes y la clasificación que arrojaba la red entrenada. Por lo mencionado anteriormente se procedió a buscar formas más optimas de ingresar los datos por lo que se decidió usar la librería sklearn de Python que entre otras cosas divide por sí misma una carpeta con enfermedades en datos para entrenamiento, validación y prueba. Esta vez, con el fin de ahorrar recursos computacionales se redimensionó las imágenes antes de ingresarlas al algoritmo, el tamaño a usar era de 21x28, una dimensión muy reducida que dependiendo de la aplicación en la que se utilice, se pueden obtener resultados bastante favorables, sin embargo, para el reconocimiento de enfermedades en hojas no es el caso. Al ingresar la base de datos el algoritmo crea dos matrices, una en la que se encuentran las imágenes y otra en donde se almacenan las etiquetas de cada clase, con el fin de mejorar la forma en que la red asimila los datos de entrada de las etiquetas

se cambió la numeración de estas a one-hot y se probó el desempeño del modelo tanto con imágenes a color como también en escala de grises donde se obtuvo mayor costo computacional, también se obtuvieron mejores resultados al trabajar con imágenes a color en RGB. Finalmente, se aumentó el tamaño de las imágenes a 256x256 y se incrementó la base de datos con la que se obtuvo resultados más consistentes en el porcentaje arrojado por el entrenamiento.

3.2. Aplicación

Se ha desarrollado un clasificador de enfermedades en plantas de tomate a través del análisis de sus hojas, el cual cuenta con dos algoritmos programados en el entorno de desarrollo integrado de Jupyter (este entorno trabaja por bloques, facilitando la obtención de resultados), haciendo uso de Python en su versión 3.6.7:

1) Programa de entrenamiento: se ha realizado con el método de redes neuronales convolucionales. En Python existen varias librerías para trabajar con redes neuronales, siendo keras en su versión 1.19.2 la escogida en este caso. Para trabajar con el conjunto de datos y con un reporte sobre la clasificación se añade la librería sklearn. Por fines prácticos se ha escogido una base de datos de la web, de imágenes de hojas de tomate con diferentes enfermedades. Previo al entrenamiento se extrajo una cantidad de imágenes para la fase de validación de resultados. Una vez realizado el ingreso, división, pre-procesamiento de los datos y matrices, se procede a crear una red conformada por funciones de activación como son: convolución, relu, max pooling; mismas que se repiten cuantas veces se crea conveniente para lograr resultados óptimos, a continuación se procede a entrenar la red.

2) Programa de clasificación: mediante la librería opencv se puede lograr la obtención de datos externos como es la recopilación de imágenes en tiempo aproximado (en este caso de hojas) haciendo uso de un teléfono celular, como cámara web e introduciendo el modelo entrenado. Además, de sus respectivos pesos y las configuraciones para que se pueda visualizar en el marco de la cámara el nombre de la predicción y la correspondiente etiqueta de la imagen.

3.2.1. Resultados y Pruebas

En esta sección se describen todos los problemas solucionados a lo largo de la creación del proyecto hasta obtener los resultados esperados del mismo.

Eficiencia del entrenador

Existen varias formas en que se pueden ingresar las imágenes al algoritmo, por lo que se realizaron múltiples cambios empezando por probar el rendimiento al ingresar imágenes en escala de grises en comparación con imágenes a color, pero esto no era suficiente para alcanzar un modelo con un margen alto de eficiencia. A continuación en la figura 3.1, se observa la cantidad de datos ingresados al sistema.

```
leyendo imagenes de  C:\Users\Gabri\Documents\TODO\TRABAJO DE GRADO I\modelo final\Pobar4origin_cel\
C:\Users\Gabri\Documents\TODO\TRABAJO DE GRADO I\modelo final\Pobar4origin_cel\Tomato___Bacterial_spot 1
C:\Users\Gabri\Documents\TODO\TRABAJO DE GRADO I\modelo final\Pobar4origin_cel\Tomato___Early_blight 560
C:\Users\Gabri\Documents\TODO\TRABAJO DE GRADO I\modelo final\Pobar4origin_cel\Tomato___Healthy 628
C:\Users\Gabri\Documents\TODO\TRABAJO DE GRADO I\modelo final\Pobar4origin_cel\Tomato___Septoria_leaf_spot 619
Directorios leidos: 4
Imagenes en cada directorio [561, 628, 619, 607]
suma Total de imagenes en subdirs: 2415
```

Figura 3.1: Directorios

Así que se trabajó en mejorar el pre-procesamiento de las imágenes (esto se consiguió trabajando con imágenes redimensionadas a un tamaño de 21x28 pixeles en RGB), como también en contribuir a que la red asimile mejor las matrices creadas para: etiquetas de cada clase y matrices con datos de pixeles de cada imagen, cambiando de la numeración decimal a one-hot, figura 3.2.

```
Original label: 2
After conversion to one-hot: [0. 0. 1. 0.]
```

Figura 3.2: Transformación de las etiqueta a one-hot.

Además, se introdujo la librería sklearn para evitar la separación manual de los datos en segmentos de entrenamiento, validación y prueba, también, con el uso de sklearn se obtuvieron los valores de eficiencia de la red al ingresarlos en el conjunto de imágenes de prueba, obteniendo así dos listas, una con los valores en donde el modelo obtuvo clasificaciones correctas y otra con las ocasiones en donde no pudo reconocer las imágenes.

Se destaca también, el uso del método "fit generator()"de la librería de keras que en comparación al método fit() contribuye a obtener un resultado más eficiente, debido a que realiza un pre-procesamiento a las imágenes, como: zoom, rotación, espejo con el fin de obtener más datos, es se logra con el método ImagenDataGenerator() para que el entrenamiento identifique

de mejor manera las enfermedades en las hojas de tomate, ya que este método analiza mejor imágenes que tengan perturbaciones.

Dentro de la red finalmente, se creó una sola capa oculta con sus respectivos comandos de activación, convolución, relu, max pooling, dropout, flaten, dense; en la figura 3.3, se muestran todas las capas.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 256, 256, 32)      896
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 256, 256, 32)      0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 128, 128, 32)      0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128, 128, 32)      0
_________________________________________________________________
flatten_2 (Flatten)          (None, 524288)            0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                16777248
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 32)                0
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 132
=================================================================
Total params: 16,778,276
Trainable params: 16,778,276
Non-trainable params: 0
_________________________________________________________________
```

Figura 3.3: Red CNN, cada fila corresponde a una capa oculta de la red.

Validación del entrenador

Una vez que se realizó la prueba de entrenamiento del modelo, que cuenta con 2415 imágenes. Como se ve en la 3.1, se ingresa la división de imágenes destinadas para este fin que corresponde al 20 % del total de la muestra, dando como resultado 483. Se obtiene en respuesta un Excel con los aciertos logrados, que son 437 y otro Excel con 46 datos incorrectos, esto se visualiza en las figuras 3.7-3.8. Además, se ingresaron imágenes directamente de la carpeta de prueba, destinado para validar el entrenamiento y el clasificador, con lo que se comprobó que funciona correctamente considerando que el porcentaje de eficiencia obtenido ha sido del 93,2 % que

Cuadro 3.1: Reporte de resultados obtenidos tras entrenar la red CNN.

Reporte de resultados				
	Precisión	Recordar	Puntaje	Apoyo
Clase 0	0,97	0,81	0,88	117
Clase 1	0,82	0,81	0,88	119
Clase 2	0,93	0,98	0,96	131
Clase 3	0,84	0,94	0,89	116
Eficiencia			0,89	483
Macro avg	0,89	0,89	0,89	483
Pesos avg	0,89	0,89	0,89	483

se puede observar en la imagen de la figura 3.6, como también, en el reporte de validación, mostrado en la figura 3.5.

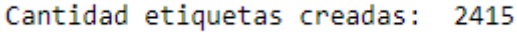

Figura 3.4: Número de etiquetas creadas.

Eficiencia del Clasificador

Se realizó un algoritmo para cargar el modelo entrenado y lograr a través de una cámara web el reconocimiento de las enfermedades, por lo cual se hizo uso de las librerías requeridas para el entrenamiento, también, de opencv que es la librería de visión artificial en Python. Primero se trabajó con la cámara web de laptop con resolución de 640x480, lo que hizo que al probar, la eficiencia de detección de como resultado porcentaje de reconocimiento casi nulo, por esta razón, se procedió a hacer uso de la cámara de un teléfono celular, con resolución de 12 Megapíxeles y se probó el reconocimiento nuevamente sin obtener los resultados esperados ,debido a que se identificó que reconocía las imágenes con las que se entreno la red, pero no reconocía imágenes nuevas, por esta razón se procedió a tomar fotografías de toda la base de datos con la cámara de teléfono celular, para que con esto las imágenes usadas para el entrenamiento del modelo, y también las adquiridas para que el clasificador reconozca, se capturen bajo los mismos parámetros, con esto se obtuvo imágenes de un tamaño de 3024x4032 píxeles, mucho mayor al que se estaba usado para el entrenamiento. En la figura 4.7 se puede visualizar una interacción correcta realizada por el clasificador .

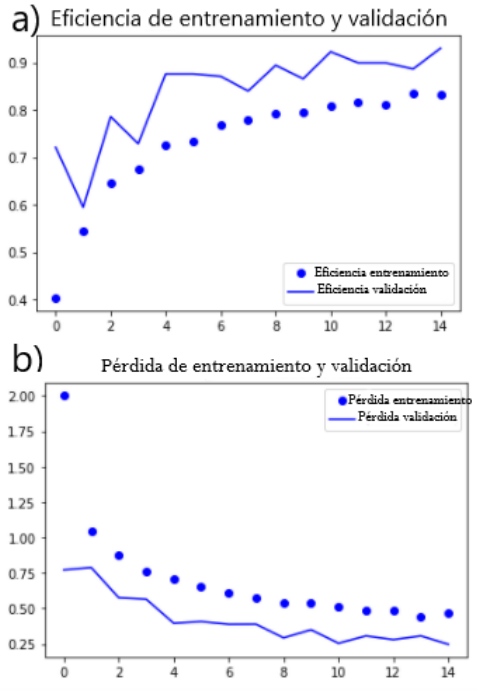

Figura 3.5: a) Curvas de eficiencia y b) pérdida del entrenamiento.

Figura 3.6: Detección de efermedades ingresando manualmente una imagen al clasificador. a) nombre de la imagen ingresada y b) clasificación realizada por el programa.

Modificaciones al entrenamiento

Por lo mencionado anteriormente, se procedió a ingresar las imágenes al entrenamiento, pero redimensionando a un tamaño de 256x256 pixeles, lo que conllevo a que el entrenamiento sea más tardado, tomando en cuenta que tomaba alrededor de 25 minutos y con el redimensionamiento de imágenes tarda aproximadamente 3 horas, se puede añadir también que la eficiencia disminuye en su porcentaje pasando de 96 % a 93 %. Tras el nuevo entrenamiento se comprobó su funcionamiento, obteniendo los resultados esperados al ingresar imágenes de la carpeta de comprobación, dando como resultado clasificaciones correctas como se puede ver en la figura 3.8.

```
WARNING:tensorflow:From C:\Users\Gabri\.conda\envs\python2\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name
tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Epoch 1/15
256/256 [==============================] - 152s 595ms/step - loss: 2.0013 - accuracy: 0.4031 - val_loss: 0.7727 - val_accuracy:
0.7209
Epoch 2/15
256/256 [==============================] - 142s 555ms/step - loss: 1.0445 - accuracy: 0.5431 - val_loss: 0.7880 - val_accuracy:
0.5943
Epoch 3/15
256/256 [==============================] - 141s 549ms/step - loss: 0.8725 - accuracy: 0.6459 - val_loss: 0.5754 - val_accuracy:
0.7855
Epoch 4/15
256/256 [==============================] - 140s 548ms/step - loss: 0.7619 - accuracy: 0.6759 - val_loss: 0.5649 - val_accuracy:
0.7287
Epoch 5/15
256/256 [==============================] - 141s 550ms/step - loss: 0.7067 - accuracy: 0.7247 - val_loss: 0.3959 - val_accuracy:
0.8760
Epoch 6/15
256/256 [==============================] - 142s 555ms/step - loss: 0.6557 - accuracy: 0.7346 - val_loss: 0.4075 - val_accuracy:
0.8760
Epoch 7/15
256/256 [==============================] - 140s 545ms/step - loss: 0.6075 - accuracy: 0.7690 - val_loss: 0.3880 - val_accuracy:
0.8708
Epoch 8/15
256/256 [==============================] - 141s 549ms/step - loss: 0.5738 - accuracy: 0.7796 - val_loss: 0.3886 - val_accuracy:
0.8398
Epoch 9/15
256/256 [==============================] - 140s 547ms/step - loss: 0.5382 - accuracy: 0.7929 - val_loss: 0.2926 - val_accuracy:
0.8941
Epoch 10/15
256/256 [==============================] - 140s 548ms/step - loss: 0.5380 - accuracy: 0.7961 - val_loss: 0.3493 - val_accuracy:
0.8656
Epoch 11/15
256/256 [==============================] - 140s 548ms/step - loss: 0.5101 - accuracy: 0.8090 - val_loss: 0.2546 - val_accuracy:
0.9225
Epoch 12/15
256/256 [==============================] - 141s 549ms/step - loss: 0.4867 - accuracy: 0.8157 - val_loss: 0.3063 - val_accuracy:
0.8992
Epoch 13/15
256/256 [==============================] - 141s 550ms/step - loss: 0.4885 - accuracy: 0.8114 - val_loss: 0.2798 - val_accuracy:
0.8992
Epoch 14/15
256/256 [==============================] - 140s 547ms/step - loss: 0.4449 - accuracy: 0.8357 - val_loss: 0.3062 - val_accuracy:
0.8863
Epoch 15/15
256/256 [==============================] - 140s 547ms/step - loss: 0.4632 - accuracy: 0.8314 - val_loss: 0.2470 - val_accuracy:
0.9302
```

Figura 3.7: Entrenamiento con 93 % de eficiencia, en donde se encuentran las filas que en cada interacción evalúan la, eficiencia, perdida, validación del entrenamiento y comprobación de la validación.

Validación del Clasificador

Se cargó el nuevo modelo entrenado y se probó su eficiencia clasificando las 4 clases de hojas de tomate: bacterial spot, early bligth, healthy, septoria del tomate; en donde se encontró,

que clasificó una gran cantidad de enfermedades con éxito. Con lo que se puede decir que fue una opción acertada el cambiar las imágenes de la base de datos por las fotografías, al hacer esto también mejoró el reconocimiento de imágenes que no fueron capturadas con la misma cámara. El modelo entrenado al ser probado ha generado ciertos datos erróneos; en el caso de las clases septoria leaf spot, healthy, da una respuesta correcta en alrededor de 19 de cada 20 imágenes, para la clase early bligth disminuyen los resultados correctos a aproximadamente 14 de cada 20 imágenes y en el caso de la clase bacterial spot encuentra coincidentes a 5 de cada 20 imágenes, estos valores han sido extraídos de someter el clasificador a las imágenes de comprobación de forma manual, tanto en la parte de ingreso de una imagen directamente de la carpeta "comprobación" al fragmento de código que realiza dicho proceso, como también a través del ingreso de imágenes mediante la cámara de teléfono celular, usada como cámara web.

Capítulo 4

Conclusiones y Recomendaciones

4.1. Conclusiones

• Después de analizar el estado del arte se logró reconocer los parámetros a considerar y los resultados que se requiere obtener, dependiendo del método a utilizar, con lo que se determina que el método más optimo para trabajar en clasificación de imágenes, dirigido a detección de anomalías en plantas, son las redes neuronales convolucionales.

• Con el desarrollo del clasificador se afirma que tanto el pre-procesamiento de los datos en que ingresan al sistema, como también el tamaño de la muestra influyen directamente con la eficiencia de los resultados alcanzados durante el entrenamiento muestra donde que a mayor cantidad de elementos, eficiencia más alta obteniendo un 93 % como el mejor resultado, con una pérdida del 0.46 % que son resultados favorables.

• El entorno en donde interactúa el clasificador puede intervenir de forma positiva si el ambiente en el que se encuentra es controlado o de forma negativa si los factores externos no son ponderados. Este factor logra variar el resultado obtenido, en este caso afectando a las clasificaciones que ejecuta el sistema, disminuyendo el porcentaje de eficiencia de este.

• Tras someter el clasificador a las imágenes destinadas para la prueba del mismo, se dice que en 3 de las clases se obtuvo resultados favorables ya que se identifican correctamente hasta 15 de cada 20 fotografías, pero en el caso de la clase bacterial spot, existe concordancia en apenas 5 de cada 20, debido a que las manchas en las hojas de dicha enfermedad son similares a las de septoria leaf spot.

4.2. Recomendaciones

4.3. Trabajo Futuro

Para el trabajo futuro de este proyecto sería importante mejorar aspectos como la eficiencia del entrenamiento, mejorar las entradas tanto como la red creada para que la red reconozca de mejor manera los rasgos característicos de las imágenes que son ingresadas y procesadas.
Se sabe también que la funcionalidad de los dispositivos electrónicos varía en su forma y tiempo de respuesta, por tal razón se recomienda una investigación sobre qué dispositivo sería más adecuado para ser usado en trabajos como el realizado.
Es fundamental una implementación del sistema en un dispositivo que sea autónomo y portátil para ser usado en cultivos directamente, en busque de contribuir a la temprana detección de enfermedades en plantas de tomate.

Bibliografía

[1] El telégrafo, «La producción del campo mejora con la tecnología,» 07 septiembre 2019. [En línea]. Available: https://www.eltelegrafo.com.ec/noticias/economía/4/producción tecnologiaecuador ministerio agricultura. [Último acceso: 14 mayo 2020].

[2] J. J. Prado, J. P. Erráez, I. Cilio , D. Godoy y N. Granizo, « Desempeño de actividades económicas seleccionadas en Ecuador periodo 2002Q1 - 2018Q3,» Boletín Macroeconónico, p. 20, 21 enero 2019.

[3] Instituto Nacional de Estadística y Censo, « Encuesta de Superficie y Producción Agropecuaria Continua» 2017. [en línea] Available: https://www.ecuadorencifras.gob.ec/documentos/webinec/Estadisticas agropecuarias/espac/espac 2017 Presentacion Principales Resultados ESPAC 2017.pdf

[4] Ruíz, M. (2007). Producción y comercialización de tomate. Quito: s/e.

[5] El Comercio, «Ocho variedades de tomate riñón están en los mercados locales,» 12 marzo 2011. [en línea] Available: https://www.elcomercio.com/actualidad/negocios/ocho variedades de tomate rinon.html

[6] Bernal Roberto, « ENFERMEDADES DE TOMATE (Lycopersicum esculentum Mill.) EN INVERNADERO EN LAS ZONAS DE SALTO Y BELLA UNIÓN,». [en línea] Available: http://www.inia.uy/Publicaciones/Documentos 20compartidos/184292307 10110412.pdf

[7] INIAP. (2008). Guía Técnica de Cultivos (Aida Villavicencio, Wilson Vásquez ed.). Quito: INIAP.

[8] Jaramillo Juan F, «Evaluación agronómica del cultivo de tomate (Solanum lycopersicum) bajo tres diferentes coberturas plásticas,» repositorio.usfq, Quito, 19 de noviembre 2015.

[9] Mantilla Tania E., «Caracterización molecular de la bacteria Clavibacter michiganensis promotora del cáncer microbiano del tomate riñón (Lycopersicon esculentum mill), en el cantón Urcuquí,» septiembre 2019. [en línea] Available: https://dspace.pucesi.edu.ec/handle/11010/546

[10] Agriculturers, «LA TECNOLOGÍA UTILIZADA EN LA AGRICULTURA,» 3 Semptiembre 2015. [en línea] Available: https://agriculturers.com/latecnologia-utilizada-en-la-agricultura/

[11] Fernández-Bregón, Noelia Urrestarazu, Miguel Valera, Diego. «Algunos usos de la visión artificial y su aplicación en la horticultura protegida,» Vida Rural 1133-8938. 343. 46-48. (2012).

[12] Ministerio de Agricultura y Ganadería, « Boletin Situacional - Tomate riñon 2017,» 5 diciembre 2018. [en línea] Available: https://fliphtml5.com/ijia/ajne/basic

[13] J. Garcia, "Plant disease identification from individual lesions and spots using deep learning", Biosystems Engineering, vol. 180, Pages 96-107, ISSN 1537-5110 (2019),[en línea]. Disponible: https://doi.org/10.1016/j.biosystemseng.2019.02.002

[14] S. Khan, A.A. Shaikh, H. Ansari,N. Ansari,"Disorder Detection in Tomato Plant Using Deep Learning". In: Ayesha, A., Ansari, H., Ansari, N. (eds.), Disorder Detection in Tomato Plant Using Deep Learning (Febrero 24, 2019) (2019), [en línea]. Disponible:https://doi.org10.1007/978-981-15-3242-919

[15] H. Sabrol y K. Satish, Çlasificación de las enfermedades de las plantas del tomate en imágenes digitales mediante el árbol de clasificación", Conferencia internacional sobre procesamiento de señales y comunicaciones (ICCSP) de 2016, Melmaruvathur, 2016, págs. 1242-1246, doi: 10.1109 / ICCSP.2016.7754351

[16] V. Maeda, C.E. Galván, L.A. Zanella, J.M. Celaya, J.I. Galván, H. Gamboa, H. Luna, R. Magallanes, C.A. Guerrero y C.A. Olvera, "Comparación de arquitecturas de redes neuronales convolucionales para la clasificación de enfermedades de plantas de tomate", Ciencias Aplicadas , vol. 10, no. 4, pág. 1245, febrero de 2020.

[17] A. Fuentes, S. Yoon, S. Kim y D. Park, "Un detector robusto basado en el aprendizaje profundo para el reconocimiento de plagas y enfermedades de plantas de tomate en tiem-

po real" , Sensores , vol. 17, no. 9, pág. 2022, septiembre de 2017 [en línea]. Disponible: http://dx.doi.org/10.3390/s17092022

[18] V. Maeda, C. Guerrero, C. A. Olvera, A. Esquivel1, G. Espinoza, R. Bordón, Redes neuronales convolucionales para la detección y clasificación de enfermedades de plantas basadas en imágenes digitales", Revista Científica Biológico Agropecuaria Tuxpan (Especial Congreso), vol. 6, pag. 3-6, noviembre 2018

[19] J.P. Salazar, E. Sánchez, R.R. Biswal, Çlasificación de clorosis en hojas de árboles de naranja mediante aprendizaje automático", Research in Computing Science, Vol. 147(5), pp. 185-195, 2018, PDF: Clasificación de clorosis en hojas de árboles de naranja mediante aprendizaje automático.

[20] J. M. F. R. Flórez, Las Redes Neuronales Artificiales, Canadá: Netbiblo, 2008.

[21] J. Bagnato, Aprende Machine Learning en Español (Teoría + Práctica Python), Leanpub, 2020.

[22] J. M. F. R. Flórez, Las Redes Neuronales Artificiales, Canadá: Netbiblo, 2008.

[23] G. Hilera, «Redes neuronales artificiales. Fundamentos, modelos y aplicaciones,» Serie Paradigma, RaMa. Autores: Raquel Flórez López, José Miguel Fernández Fernández., 1994.

[24] A. P. Carrasco, «Clasificación de imágenes usando redes neuronales,» Dpto. Teoría de la Señal y Comunicaciones, Universidad de Sevilla, vol. 1, nº 1, p. 28, 2019.

Apéndice

4.4. Código del entrenador CNN

```
import numpy as np                              #librería para creación de vectores y matrices
import os
import re
import matplotlib.pyplot as plt                 #librería de generación de gráficos
%matplotlib inline
from sklearn.model_selection import train_test_split  #librería para aprendizaje automático
from sklearn.metrics import classification_report
from skimage.transform import resize
```

```
import keras                                    #librería de redes neuronales
from keras.utils import to_categorical
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
from keras.preprocessing.image import ImageDataGenerator
```

```
#recepción de la información
dirname = os.path.join(os.getcwd(), 'Pobar4origin_cel')
imgpath = dirname + os.sep

images = []
directories = []
dircount = []
prevRoot=''
cant=0

print("leyendo imagenes de ",imgpath)

for root, dirnames, filenames in os.walk(imgpath):
    for filename in filenames:
        if re.search("\.(JPG|jpg|jpeg|png|bmp|tiff)$", filename):
            cant=cant+1
            filepath = os.path.join(root, filename)
            imagenes = plt.imread(filepath)
            #imagenes = cv2.cvtColor(imagenes, cv2.COLOR_BGR2GRAY)# m2
            #imagenes = cv2.equalizeHist(imagenes) # m2
            imagenes= resize(imagenes, (256, 256),anti_aliasing=True,clip=False,preserve_range=True)
            images.append(imagenes)
            b = "cargando..." + str(cant)
            print (b, end="\r")
            if prevRoot !=root:
                print(root, cant)
                prevRoot=root
                directories.append(root)
                dircount.append(cant)
                cant=0
dircount.append(cant)

dircount = dircount[1:]
dircount[0]=dircount[0]+1
print('Directorios leidos:',len(directories))
print("Imagenes en cada directorio", dircount)
print('suma Total de imagenes en subdirs:',sum(dircount))
```

```
#creación de etiquetas
labels=[]
indice=0
for cantidad in dircount:
    for i in range(cantidad):
        labels.append(indice)
    indice=indice+1
print("Cantidad etiquetas creadas: ",len(labels))
```

```
#Da el número de la clase que corresponde a cada enfermedad
enfermedades=[]
indice=0
for directorio in directories:
    name = directorio.split(os.sep)
    print(indice , name[len(name)-1])
    enfermedades.append(name[len(name)-1])
    indice=indice+1
# Las etiquetas aqui creadas sirven para el programa de vision artificial
```

```
y = np.array(labels)
X = np.array(images, dtype=np.uint8) #convierto de lista a numpy

# toma solo los numeros diferentes de entre todas las etiquetas
classes = np.unique(y)
nClasses = len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)
```

```
#divisón de la muestra total en datos de entrenamiento y prueba
train_X,test_X,train_Y,test_Y = train_test_split(X,y,test_size=0.2)
print('Training data shape : ', train_X.shape, train_Y.shape)
print('Testing data shape : ', test_X.shape, test_Y.shape)
```

```
#comprobación del correcto ingreso de las imagenes
plt.figure(figsize=[5,5])

# Muestra la primera imagen en los datos de entrenamiento
plt.subplot(121)
plt.imshow(train_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(train_Y[0]))

# Muestra la primera imagen en los datos de prueba
plt.subplot(122)
plt.imshow(test_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(test_Y[0]))
```

```
# Convertir a coma flotante(decimales) y normalizar (0-1)
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255.
test_X = test_X / 255.
```

```
# Cambiar las etiquetas de codificación categórica a one-hot
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

# Muestra el cambio para la etiqueta de categoría usando codificación one-hot
print('Original label:', train_Y[0])
print('After conversion to one-hot:', train_Y_one_hot[0])
```

```
#Mezclar todo y crear los grupos de entrenamiento y testing
train_X,valid_X,train_label,valid_label = train_test_split(train_X, train_Y_one_hot, test_size=0.2, random_state=13)
```

```
print(train_X.shape,valid_X.shape,train_label.shape,valid_label.shape)
```

```
# Genera variaciones en la imagen a entrenar para que la red sea capaz
#de adaptarse a distintas condiciones de imagenes
augmenter = ImageDataGenerator(horizontal_flip=True,
                               rotation_range=0.15,
                               width_shift_range=0.1,
                               height_shift_range=0.1,
                               zoom_range=0.15,
                               shear_range=0.1,
                               fill_mode='nearest')
```

```
#declaramos variables con los parámetros de configuración de la red
INIT_LR = 1e-3 # Valor inicial de learning rate. El valor 1e-3 corresponde con 0.001
epochs = 15 # Cantidad de iteraciones completas al conjunto de imagenes de entrenamiento
batch_size = 10 # cantidad de imágenes que se toman a la vez en memoria
```

```
diseases_model = Sequential()
diseases_model = Sequential()
diseases_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',padding='same',input_shape=(256,256,3)))
diseases_model.add(LeakyReLU(alpha=0.1))
diseases_model.add(MaxPooling2D((2, 2),padding='same'))
diseases_model.add(Dropout(0.5))

diseases_model.add(Flatten())
diseases_model.add(Dense(32, activation='linear'))
diseases_model.add(LeakyReLU(alpha=0.1))
diseases_model.add(Dropout(0.5))
diseases_model.add(Dense(nClasses, activation='softmax'))
```

```
diseases_model.summary()
```

```
diseases_model.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer=keras.optimizers.Adagrad(lr=INIT_LR,
                        decay=INIT_LR / 100),metrics=['accuracy'])
```

```
# Entrenador sin efectos en la imagen.
# Nota: La unica diferencia con el otro metodo de entrenamiento es que este metodo no
#hace modificaciones en las imagenes y las entrena tal cual.
diseases_train = diseases_model.fit(train_X,
                                    train_label,
                                    batch_size=batch_size,
                                    epochs=epochs,
                                    verbose=1,
                                    validation_data=(valid_X, valid_label))
```

```
# Entrenador con data augmenter (efectos en la imagen). Usar con datagenerator
# Nota: Este es metodo es más util para detectar imagenes que puedan venir con distorsiones
diseases_train = diseases_model.fit_generator(augmenter.flow(train_X, train_label, batch_size=batch_size),
                                    steps_per_epoch = 256,
                                    epochs=epochs,
                                    verbose=1,
                                    validation_data=(valid_X, valid_label))
```

```
# guardamos la red, para reutilizarla en el futuro, sin tener que volver a entrenar
diseases_model.save("modelo final_.h5py")
```

```
# evaluación de resultados
test_eval = diseases_model.evaluate(test_X, test_Y_one_hot, verbose=1)
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
# graficación de los resultados arrojados por la red entrenada
accuracy = diseases_train.history['accuracy']
val_accuracy = diseases_train.history['val_accuracy']
loss = diseases_train.history['loss']
val_loss = diseases_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

```
# graficación de los resultados arrojados por la red entrenada
accuracy = diseases_train.history['accuracy']
val_accuracy = diseases_train.history['val_accuracy']
loss = diseases_train.history['loss']
val_loss = diseases_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

```
predicted_classes2 = diseases_model.predict(test_X)
```

```
predicted_classes=[]
for predicted_diseases in predicted_classes2:
    predicted_classes.append(predicted_diseases.tolist().index(max(predicted_diseases)))
predicted_classes=np.array(predicted_classes)
```

```
predicted_classes.shape, test_Y.shape
```

```
correct = np.where(predicted_classes==test_Y)[0]
print("Found %d correct labels" % len(correct))
filenames = []
preditsdata = []

for i, correct in enumerate(correct[0:len(correct)]):
    #plt.subplot(3,3,i+1)
    #plt.imshow(test_X[correct].reshape(26,21,3), cmap='gray', interpolation='none')
    #plt.title("{}, {}".format(enfermedades[predicted_classes[correct]],
                                                    #enfermedades[test_Y[correct]]))
    filenames.append(enfermedades[predicted_classes[correct]])
    preditsdata.append(enfermedades[test_Y[correct]])
    plt.tight_layout()

print (i)
print (filenames)
print (preditsdata)
```

```
incorrect = np.where(predicted_classes!=test_Y)[0]
print("Found %d incorrect labels" % len(incorrect))

filenames1 = []
preditsdata1 = []

for i, incorrect in enumerate(incorrect[0:len(incorrect)]):
    #plt.subplot(3,3,i+1)
    #plt.imshow(test_X[incorrect].reshape(26,21,3), cmap='gray', interpolation='none')
    #plt.title("{}, {}".format(enfermedades[predicted_classes[incorrect]],
                                                    #enfermedades[test_Y[incorrect]]))
    filenames1.append(enfermedades[predicted_classes[incorrect]])
    preditsdata1.append(enfermedades[test_Y[incorrect]])
    plt.tight_layout()

print (i)
print (filenames1)
print (preditsdata1)
```

```
import pandas as pd
target_names = ["Class {}".format(i) for i in range(nClasses)]
print(classification_report(test_Y, predicted_classes, target_names=target_names))

results = pd.DataFrame({"Filename":filenames,
                        "Predictions":preditsdata})
results.to_csv("correct.csv", index=False,sep = ';')

results = pd.DataFrame({"Filename":filenames1,
                        "Predictions":preditsdata1})
results.to_csv("incorrect.csv", index=False,sep = ';')
```

4.5. Código del Clasificador

```
#Importación de librerías usadas para el entrenador
import keras
from keras.utils import to_categorical
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
import cv2
import numpy as np
import os
import re
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from skimage.transform import resize
#import pandas
```

```
# Cargar el modelo entrenado y el indice de enfermedades segun el entrenamiento
path_model = 'modelo final_.h5py' # Nombre del modelo entrenado
enfermedades = [
'Tomato___Bacterial_spot',
'Tomato___Early_blight',
'Tomato___healthy',
'Tomato___Septoria_leaf_spot'
] # Todas las enfermedades con las que se entreno la red

nClasses = len(enfermedades)

# Modélo de la red empleada para entrenar
diseases_model = Sequential()
diseases_model = Sequential()
diseases_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',padding='same',input_shape=(256,256,3)))
diseases_model.add(LeakyReLU(alpha=0.1))
diseases_model.add(MaxPooling2D((2, 2),padding='same'))
diseases_model.add(Dropout(0.5))

diseases_model.add(Flatten())
diseases_model.add(Dense(32, activation='linear'))
diseases_model.add(LeakyReLU(alpha=0.1))
diseases_model.add(Dropout(0.5))
diseases_model.add(Dense(nClasses, activation='softmax'))

diseases_model.load_weights(path_model) # Carga los pesos de la red entrenada (Literalmente cargar la red entrenada)
```

```python
# Configuracion de la cámara
frameWidth= 720          # 720 pixeles ancho
frameHeight = 1080       # 1080 pixeles alto
brightness = 60          # Brillo

font = cv2.FONT_HERSHEY_SIMPLEX

# Establecer la configuración de la cámara de video
cap = cv2.VideoCapture('https://192.168.1.3:4343/video')
# Direccion de la camara web empleada por el celular
cap.set(3, frameWidth)
cap.set(4, frameHeight)
cap.set(10, brightness)
```

```python
# Codigo para vision artificial
threshold = 0.80        # PROBABLITY THRESHOLD
while True:

    # READ IMAGE
    images=[]
    success, imgOrignal = cap.read()
    # Show image Original
    cv2.imshow("Processed Image", imgOrignal)
    # PROCESS IMAGE (Aplicar las conversiones necesarias a la imagen para que esten en formato adecuado)
    image_resized = resize(imgOrignal, (256, 256),anti_aliasing=True,clip=False,preserve_range=True) # redimensionar a 256x256 RG
    images.append(image_resized) # Agregar imagen al array para quedar con forma [1,256,256,3]
    images = np.array(images, dtype=np.uint8) #convierto de lista a numpy
    images = images.astype('float32') # Soporte a decimal
    images = images / 255.    # Normalizar (0-1)

    predicted_classes = diseases_model.predict(images) # Pos aplicar la prediccion
    lista  = predicted_classes.tolist() # Convertir a lista la prediccion para extraer sus caracteristicas
    probabilityValue = max(lista[0]) # Extrae el valor de probabilidad correspondiente a la enfermedad con mayor probabilidad de
    classIndex = lista[0].index(probabilityValue) # Extraer el indice que corresponde a la enfermedad más probable

    if probabilityValue > threshold:
        #print(enfermedades[classIndex])
        cv2.putText(imgOrignal,str(classIndex)+" "+str(enfermedades[classIndex]), (120, 35), font, 0.75, (0, 0, 255), 2, cv2.LINE
        cv2.putText(imgOrignal, str(round(probabilityValue*100,2) )+"%", (180, 75), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)
        cv2.imshow("Result", imgOrignal)

    if cv2.waitKey(1) and 0xFF == ord('q'):
```

```python
# Codigo para probar imagenes individuales
images=[]
# se especifica la imagen que se ingresará para probarla
filenames = ['test/bacterial_spot 9.jpg']

for filepath in filenames:
    image = plt.imread(filepath,0)
    image_resized = resize(image, (256, 256),anti_aliasing=True,clip=False,preserve_range=True)
    images.append(image_resized)

X = np.array(images, dtype=np.uint8) #convierto de lista a numpy
test_X = X.astype('float32')
test_X = test_X / 255.

predicted_classes = diseases_model.predict(test_X)

for i, img_tagged in enumerate(predicted_classes):
    print(img_tagged.tolist())
    print(filenames[i], enfermedades[img_tagged.tolist().index(max(img_tagged))])
```