Desarrollo de una herramienta computacional basada en redes neuronales para el diagnóstico del tizón tardío en cultivos de papa

Camilo Andrés Ortiz Daza

Universidad Antonio Nariño

Facultad de Ingeniería Mecánica, Electrónica y Biomédica

Ciudad, Colombia

2021

Desarrollo de una herramienta computacional basada en redes neuronales para el diagnóstico del tizón tardío en cultivos de papa

Camilo Andrés Ortiz Daza

Proyecto de grado presentado como requisito parcial para optar al título de:

Magister en Instrumentación y Automatización

Director:

(Ph.D.) Christian Camilo Erazo Ordoñez

Línea de Investigación:

Procesamiento digital de imágenes y señales.

Grupo de Investigación:

Grupo de Investigación en Bioinstrumentación y Control (GIBIO)

Universidad Antonio Nariño

Facultad de Ingeniería Mecánica, Electrónica y Biomédica

Bogotá, Colombia

2021

(Dedicatoria o lema)

A mis padres y a mi adorada esposa que siempre han estado para apoyarme…

Resumen

El tizón tardío (Phythottora Infestas) es una enfermedad que afecta seriamente a los cultivos de papa, propiciando un impacto negativo en la economía del agricultor. Este proyecto permitió construir una herramienta computacional llamada NeuroPI – 2105 basada en una red neuronal convolucional creada por el autor, misma que clasifica dos tipos de folios que son; sanos y enfermos. La red ha sido entrenada con 1304 imágenes de la base de datos PlantVillage, 304 de ellas corresponden a hojas sanas, siendo el restante atribuidas al tizón tardío. El algoritmo de entrenamiento ha empleado el gradiente descendente Adam, la función del error de entropía cruzada y la backpropagation, con el fin de ajustar los pesos sinápticos y los niveles de umbral en la red.

Lo anterior, ha sido implementado en Matlab versión 2018, añadiendo un filtro espacial paso bajo para el tratamiento de imágenes contaminadas, así como una función para visualizar los artefactos que ella posee. Se ha obtenido una adecuada organización en la base de datos dividida en una relación $80 / 20$, conformando los conjuntos de entrenamiento y prueba en la fase de aprendizaje. Asimismo, al aplicar pruebas de repetitividad con 500 imágenes distintas la NeuroPI – 2105 es capaz de identificar hojas enfermas presentando un error de -2.8 imágenes sin filtro y nulo con tratamiento digital. La red alcanzó una exactitud del 99.18%, se recomienda usar el filtro para el diagnóstico de hojas enfermas porque disminuye la tasa de error, ofreciendo un intervalo de confianza del 95.4% que esta entre $[497.633, 502.367]$.

Palabras clave: Tizón Tardío, Herramienta Computacional, Redes Neuronales Convolucionales, Matlab.

Abstract

Late blight (Phythottora Infestas) is a disease that seriously affects potato crops, causing a negative impact on the farmer's economy. This project will generate a computational tool called NeuroPI - 2105 based on a convolutional neural network created by the author, which classifies two types of leafs that are; healthy and sick. The network has been trained with 1304 images from the PlantVillage database, 304 of them correspond to healthy leaves, the remainder being attributed to late blight. The training algorithm has used the Adam gradient descent, the cross-entropy error function, and backpropagation, in order to adjust the synaptic weights and threshold levels in the network.

The above has been implemented in Matlab version 2018, adding a low-pass spatial filter for the treatment of contaminated images, as well as a function to visualize the artifacts that it has. An adequate organization has been obtained in the database divided into an 80/20 ratio, conforming the training and test sets in the learning phase. Likewise, when applying repeatability tests with 500 different images, the NeuroPI - 2105 is able to identify diseased leaves presenting an error of -2.8 images without filter and null with digital treatment. The network reached an accuracy of 99.18%, it is recommended to use the filter for the diagnosis of diseased leaves because it reduces the error rate, offering a confidence interval of 95.4% that is between [497.633,502.367].

Keywords: Late Blight, Computational Tool, Convolutional Neural Networks, Matlab.

Contenido

Lista de figuras

Pág.

Lista de tablas

Pág.

Introducción

En Colombia el sector agricultor ha sido considerado como uno de los más esperanzadores para contribuir al desarrollo económico del país [1], siendo el cultivo de papa uno de los más relevantes. El tubérculo es comercializado en su estado fresco en un 90%, el 10% restante es adquirido por la industria de procesamiento para el consumo humano. Su cadena productiva genera anualmente cerca de 264.000 empleos; 75.000 directos y 189.000 indirectos, lo cual beneficia en promedio a 100.000 familias, en 10 departamentos y 283 municipios [2]. Además, sus cultivos proveen a la gente vinculada a este sector de grandes oportunidades laborales y económicas, lo cual implica que el 90% del área cultivable se atribuya principalmente a cuatro departamentos que son; Cundinamarca con el 37%, Boyacá con el 27%, Nariño con el 20% y Antioquia con el 6% [3].

Las enfermedades más comunes que se exhiben en los cultivos de papa son: Tizón Tardío o gota (Phytophthora Infestans), Tizón Temprano (Alternaria Solani), Rhizoctonia – Costa Negra (Rhizoctonia Solani), Roya Común (Puccinia Pittieriana), Cinicilla o Mildeo Polvoso (Erysiphe Sichoracearum), Mortaja Blanca o Palomillo (Rosellinia Sp), Marchitez Bacteriana (Rastolnia Solanacearum), Pata Negra (Erwinia Carotovora), Virus del Enrollamiento de las Hojas (PLRV), Mop Top (PMTV) y Virus de Amarillamiento de las Venas (PYVV) [4], [5] y [6], siendo las dos primeras, las afecciones más graves que limitan la productividad del tubérculo [7] y [8]. Lo anterior ha reflejado un gran impacto económico que está en función de dicha productividad, obligando a un alza en los precios de comercialización para su consumo [9].

Asimismo, las condiciones estacionales como el cambio climático originan que estas siembras sufran ataque de pestes o morbilidades como el Tizón Tardío y Tizón Temprano, ellas actúan primero sobre las hojas y después en toda la planta. De este modo, se afecta la calidad y cantidad del tubérculo, por ello, al no efectuar una detección temprana de estos patógenos pueden acarrear serias pérdidas monetarias en el sector papero [10] y [11].

Sumado a ello, algunos agricultores carecen del conocimiento apropiado sobre dichas enfermedades así como la existencia de curas o tratamientos para combatirlas [10], acarreando un problema en los sectores agro – productivos más humildes. De este modo, un adecuado control, tratamiento, así como un temprano diagnóstico de las enfermedades anteriormente descritas constituyen el mayor reto para el sector papero.

Dada esta preocupación, varias investigaciones han surgido, ello con el fin desarrollar y emplear herramientas que involucren redes neuronales para la detección temprana de peste o enfermedades que afecten los cultivos. Es el caso de [12], [13], [14] y [15], quienes usan la base de datos *PlantVillage* también usada en este proyecto. En [12], se presentó una herramienta llamada *"VegeCare"* para evaluar el crecimiento de las hojas, diagnosticar patógenos en las hojas de papa y detectar el tipo de peste de la planta. Entrenan una Convolutional Neural Network (CNN) para predecir tres clases de hojas que son; sanas, enfermas de Tizón Temprano y de Tizón Tardío. Concluyen que la exactitud es superior al 96% para tareas de clasificación de hojas sanas y enfermas.

En [13], conformaron una base de datos a partir de *PlantVillage* usado la técnica de aumentado de datos, utilizando una CNN para clasificar hojas sanas y enfermas con Tizón Tardío y con Tizón Temprano en cultivos de papa, evaluándola con 50 imágenes, obteniendo una exactitud del 98%. Ellos concluyen que la red clasifica en un 99.4% en la fase de entrenamiento y 98% en la fase de validación. En [16], trabajaron con una base de datos que contiene 87.848 fotografías de hojas sanas e infectadas siendo dividida a razón de 80 / 20 para entrenamiento y validación que es la más común en aplicaciones con CNNs. Dicha base se usó para entrenar y probar cinco CNN: AlexNet, AlexNetOWTBn, GoogleNet, OverFeat, VGGNet, conformando así 58 clases en su capa de salida. El autor identifica que la red no identifica correctamente 82 imágenes de 17.548, concluyendo que estas CNN son altamente adecuadas para el diagnóstico adecuado de enfermedades en las plantas por medio del análisis de sus hojas.

En [14], usan técnicas de pre – procesamiento tales como reformado, re – escalamiento, y conversión a formato de matrices, usando 20 clases de clasificación. Usan la CNN AlexNet, la cual distingue distintos tipos de clases de 38 plantas de cultivo, la cual obtuvo una exactitud mayor que el 96.50%. Por último, han desarrollado una aplicación para identificar cada una de las clases para lo cual la red fue entrenada. En [15], entrenan una CNN que

permita clasificar enfermedades en cultivos de tomate, han construido una interfaz web de usuario empleando una arquitectura VGG para la detección de enfermedades de las hojas de tomate, entre ellas, Tizón Temprano y Tizón Tardío, alcanzando una exactitud del 96.87% en la fase de entrenamiento y de 93.124% en la fase de validación.

No obstante, en [17] se ha presentado un sistema basado en visión artificial usando procesamiento de imágenes y Support Vector Machine (SMV) para el diagnóstico de la enfermedad en las hojas usando misma base de datos de este proyecto, obteniendo una exactitud del 95%. En [18], tomaron el mismo repositorio para presentar un sistema de identificación automática basado en el procesamiento de imágenes logrando un nivel de confianza del 96% en la clasificación. En [19], se ha desarrollado un sistema que usa la clasificación y detección de las enfermedades en las hojas de las plantas usando una CNN alcanzando un 98.29% de certeza en la clasificación. En [20], se elaboró un sistema automatizado basado en CNN para predecir las enfermedades de la papa con el fin de ayudar a los cultivadores en la toma de decisiones con una confiabilidad del 98.33%.

En el presente trabajo se ha obtenido una división de 80 / 20 en la base de datos para entrenamiento y prueba respectivamente, un comportamiento dinámico de los pesos y un nivel de exactitud del 99.18% en la red neuronal construida. La herramienta computacional se sometió a 20 pruebas de repetitividad, donde se observó que el menor error producido fue de magnitud nula con una incertidumbre de medición de 2.367, usando 500 imágenes de prueba, en la clasificación de imágenes con Tizón Tardío utilizando un filtro espacial con elemento estructurante de radio unitario, en consecuencia, el máximo error encontrado corresponde a la detección de hojas sanas empleando el mismo filtro. Se concluye que dicha herramienta permite clasificar imágenes de hojas sanas y de Tizón Tardío, se recomienda usar para el diagnóstico de hojas enfermas puesto que para hojas sanas la herramienta no aprueba el ensayo de calificación, por lo cual se sugieren las recomendaciones del caso.

Por último, el autor ha conformado una base de datos propia, las imágenes han sido tomadas en los corregimientos de Pasquilla y Mochuelo Alto. Sin embargo, dicha base de datos no pertenece al conjunto de elementos de entrenamiento de la red neuronal convolucional, puesto que, su propósito solo está orientado a observar la funcionalidad de la NeuroPI – 2105.

1. Introducción al problema

Este capítulo esboza la descripción del planteamiento del problema haciendo una breve revisión sobre la importancia del cultivo de papa en Colombia, su ciclo fenológico, así como las plagas y enfermedades más comunes que los afecta, haciendo énfasis en los desafíos que enfrenta el agricultor cuando hay presencia del Phytophthora Infestans de Bary, productor del Tizón Tardío o gota en las siembras de papa.

1.1 Los cultivos de papa en Colombia

En Colombia la papa (Solarum Tuberosum) es uno de los alimentos con mayor impacto en el país [21], pues el 90% de este producto es comercializado en estado fresco y el 10% restante es aprovechado por la industria de procesamiento [3], ocupando así, el cuarto lugar de consumo a nivel mundial [7]. Además, sus cultivos proveen a la gente vinculada al sector papero de grandes oportunidades laborales y económicas, lo cual implica que el 90% del área cultivable se atribuya principalmente a cuatro departamentos que son; Cundinamarca con el 37%, Boyacá con el 27%, Nariño con el 20% y Antioquia con el 6% [3].

Lo anterior no solo beneficia al sector rural, sino que también a la economía del país, gracias a la atracción turística que disfruta de la gastronomía típica colombiana. No obstante, se evidencia un desaprovechamiento del terreno colombiano puesto que de las 22 millones de hectáreas preparadas para la siembra del tubérculo sólo 4.8 millones de ellas están siendo cultivadas, conllevando a buscar estrategias que permitan mejorar sustancialmente los procesos agro – productivos para que sean más eficientes; ello implica un enorme desafío en la trasformación de este sector, que deberá ser encaminado a apoyarse en las tecnologías para agricultura de precisión, automatización en la manutención o cuidado de la siembra, instrumentación aplicada, entre otros [1].

Actualmente, la producción de la papa aporta 264.000 empleos, 75.000 de ellos directos y 189.000 indirectos, beneficiando en promedio a 100.000 familias, de 10 departamentos y 283 municipios [2]. De acuerdo con la Food and Agriculture Organization (FAO), al fortalecer dicha producción, el país puede llegar a convertirse en una de las más grandes despensas del mundo y en uno de los siete países latinoamericanos con alto potencial para el desarrollo en áreas cultivables [1].

1.2 Proceso productivo del cultivo

Dichas áreas deben cumplir con las siguientes condiciones: temperatura ambiente entre 12 a 14 grados Celsius (˚C), humedad relativa (HR) entre 75 y 80%, pH del suelo entre 5.5 a 7.0, requerimiento hídrico entre 600 a 800 milímetros (mm) al año, tipo de suelo “franco” con pendiente máxima del 30%, entre otros [4]. Las condiciones ambientales varían con la altura del terreno, los suelos deben ser de textura fina, porosidad del 50% y una distribución equilibrada de macroporos, mesoporos y microporos, para garantizar el nivel óptimo de agua y aireación en la zona radicular de la planta [4].

Asimismo, el pH determina el nivel de acides o alcalinidad del suelo, esta medida es adimensional y se basa en la concentración de iones de hidrogeno. La letra “p” viene de la palabra potencia mientas que “H” es el símbolo empleado para denotar el elemento hidrogeno, por lo tanto, al unirlos resulta pH y equivale al exponente de los iones de hidrogeno contenidos en el suelo [22]. Sí el pH del terreno es inferior a 5.5 puede existir un aumento en los niveles de aluminio y magnesio siendo tóxico para la siembra de papa, análogamente, sí es superior a 7.5 hay una limitación en la absorción de hierro, magnesio y zinc por parte de la planta [4]. Una medida equivalente a 7.0 conlleva a un nivel neutral y sirve como punto de referencia en la escala de pH, así, bajos niveles traducen altos indicadores de acides mientras que medidas altas de pH elevan la alcalinidad del terreno.

Por lo tanto, las hectáreas destinadas para la siembra de papa que cumplan con todas las condiciones anteriormente descritas propician el éxito en la plantación, trayendo consigo un ciclo fenológico el cual contiene cuatro etapas, ellas son: plántula, desarrollo, crecimiento, tuberculización y producción, y madurez fisiológica. Cada ciclo aporta al

desarrollo botánico y morfológico de la planta, y está segmentado en dos partes llamadas subterránea y aérea. La primera atañe el crecimiento profundo de la planta; es decir, el aumento de la raíz, los estolones y los tubérculos, esto a partir del tubérculo madre. La segunda parte está ligada a la germinación y conformación de los tallos principales y secundarios, las hojas, flores y frutos, muriendo una vez finalizado el ciclo de producción. Lo anterior representa el crecimiento vegetativo, así como el reproductivo concierne a la germinación, crecimiento y desarrollo de los tubérculos, implicando que el cierre del cultivo sea contemplado entre 35 a 40 días después de la emergencia de la papa [4, p. 12].

1.3 Plagas y enfermedades en los cultivos de papa

A menudo, la siembra de papa despliega una gran variedad de plagas y enfermedades en la plantación que involucran complejas interacciones entre la planta huésped, el virus y su vector, estableciendo así un tipo de morbilidad o peste especifico, llevando al mayor desafío en el sector agrícola [23]. En Colombia las plagas que afectan con frecuencia a los tubérculos y las plantas están clasificadas en tres tipos, que son: plagas de suelo/tubérculo, del follaje y/o frutos, y de almacenamiento del producto [4].

Las enfermedades más comunes que se exhiben en este tipo de cultivo son: Tizón Tardío o gota (Phytophthora Infestans), Tizón Temprano (Alternaria Solani), Rhizoctonia – Costa Negra (Rhizoctonia Solani), Roya Común (Puccinia Pittieriana), Cinicilla o Mildeo Polvoso (Erysiphe Sichoracearum), Mortaja Blanca o Palomillo (Rosellinia Sp), Marchitez Bacteriana (Rastolnia Solanacearum), Pata Negra (Erwinia Carotovora), Virus del Enrollamiento de las Hojas (PLRV), Mop Top (PMTV) y Virus de Amarillamiento de las Venas (PYVV) [4], [5] y [6].

Este trabajo se centrará en una enfermedad en particular, el Tizón Tardío. Este patógeno es reportado como uno de los más limitantes e importantes siendo su agente causante el Oomicete Phytophthora Infestans de Bary [24], afectando principalmente las hojas, los tallos y los tubérculos. En las hojas; la enfermedad nace revelando marchas irregulares de color verde pálido a oscuro y son formadas en los bordes y ápice de los folios, conllevando a amplias lesiones necróticas de color marrón y negro rodeadas de un halo amarillento, en los tallos; las lesiones son oscuras continuas que alcanzan hasta 10 centímetros de

longitud; finalmente, en el tubérculo aparecen depresiones irregulares de textura dura, así al levantar su piel o al hacer cortes transversales se estima una necrosis de color marrón [5] y [6].

1.4 Descripción del problema

En algunas ocasiones, el área cultivada abarca gran cantidad de hectáreas entrañando una difícil labor al momento de identificar enfermedades en la siembra de papa para mantenerlas sanas y libres de plagas. Sumado a ello, algunos agricultores carecen del conocimiento apropiado sobre las enfermedades o las curas existentes para cada una de ellas [10], lo cual plantea un problema en los sectores agro – productivos más humildes. Los cultivos de papa son infectados con hongos propiciando a los dos tipos de infecciones más graves que son: Alternaria Solani y Phytophthora Infestans [7] y [8]. De este modo, el adecuado control, tratamiento, así como el temprano diagnóstico de estas enfermedades constituyen el mayor reto para el sector papero.

En afinidad con F. Islam et Al, (…) “las enfermedades en las plantas es una causa común de la perdida de producción de cultivos los cuales tendrán un notable impacto económico derivando en una reducción de producción y distribución, así como en un alza de precios para los consumidores” (…) [9, p. 127]. Asimismo, las condiciones estacionales como el cambio climático, originan que las siembras de papa sufran ataque de pestes o morbilidades como las anteriormente mencionadas, actuando primero sobre las hojas y después en toda la planta. De este modo, se afecta la calidad y cantidad del tubérculo; por lo tanto, la no detención de estos patógenos pueden acarrear serias pérdidas monetarias para el sector papero [10] y [11]

Un mecanismo muy usado por los agricultores para abordar esta problemática es llevar a cabo un minucioso diagnostico a mano de expertos, que a veces incluye análisis de laboratorio para la detección temprana de enfermedades en las plantas sembradas, aun así, ello está sujeto a juicios que en algunos casos pueden ser erróneos por diversos factores tales como; falsas lecturas en la apreciación de las hojas, errores cognitivos, falta de experticia, entre otros [25] y [26]. A su vez, las pruebas de laboratorio pueden ser

numerosas, tediosas y costosas, afectando sobre todo a aquellos papi – cultores con escases de recursos económicos [25].

Por consiguiente, este trabajo de grado se concentra en el desarrollo de una herramienta computacional que permita el diagnóstico del Tizón Tardío para atender a la problemática anteriormente descrita, usando una Covolutional Neural Network (CNN); la cual, es entrenada mediante una base de datos de hojas sanas y con Tizón Tardío para obtener un nivel aceptable del reconocimiento de las características propias de la enfermedad, basada en las técnicas de procesamiento digital de imágenes.

1.5 Objetivos del trabajo

Dado lo anterior, el trabajo de grado (modalidad – profundización) dará alcance a los siguientes objetivos:

1.5.1 Objetivo general

Construir una herramienta computacional basada en una red neuronal que permita el diagnóstico de hojas enfermas de tizón tardío en cultivos de papa.

1.5.2 Objetivos específicos

- Agrupar el conjunto de datos de hojas sanas e infectadas con el tizón tardío.

- Implementar una CNN para el tratamiento y clasificación de imágenes de hojas sanas y enfermas con tizón tardío.

- Diseñar una herramienta informática (software) para el diagnóstico de tizón tardío en hojas de plantas de papa usando Matlab R2018, Python u otros entornos de programación.

2. Marco teórico

En este capítulo se abordan las bases teóricas del aprendizaje profundo partiendo del concepto de una neurona. Su objetivo es introducir al lector en el funcionamiento y arquitectura de las redes neuronales artificiales, así como en los algoritmos de entrenamiento más usados. Se define el modelo matemático de una neurona artificial como un sistema de entrada – salida, esta última, es originada gracias a la función de activación aplicada a la combinación lineal que yace del nodo suma, así, la arquitectura de una red neuronal surge de las interconexiones entre estos nodos neuronales.

Se ha estudiado el perceptrón de Rosenblat como clasificador binario para mostrar como una red neuronal clasifica las características de la entrada, ello fija unas reglas de decisión, permitiendo establecer el teorema de convergencia del perceptrón. A partir del modelo de Rosenblat se describe algoritmo del perceptrón por lotes, las funciones de activación también son estudiadas resaltando las más usadas en redes neuronales, se introduce el concepto de redes multicapa, redes neuronales convolucionales, así como los algoritmos para entrenarlas.

2.1 Fundamentos de redes neuronales

A continuación, se estudian dos conceptos básicos de las redes neuronales que son: el modelo de la neurona artificial y los tipos de arquitectura basados en la integración de las mismas.

2.1.1 Modelo de la neurona artificial

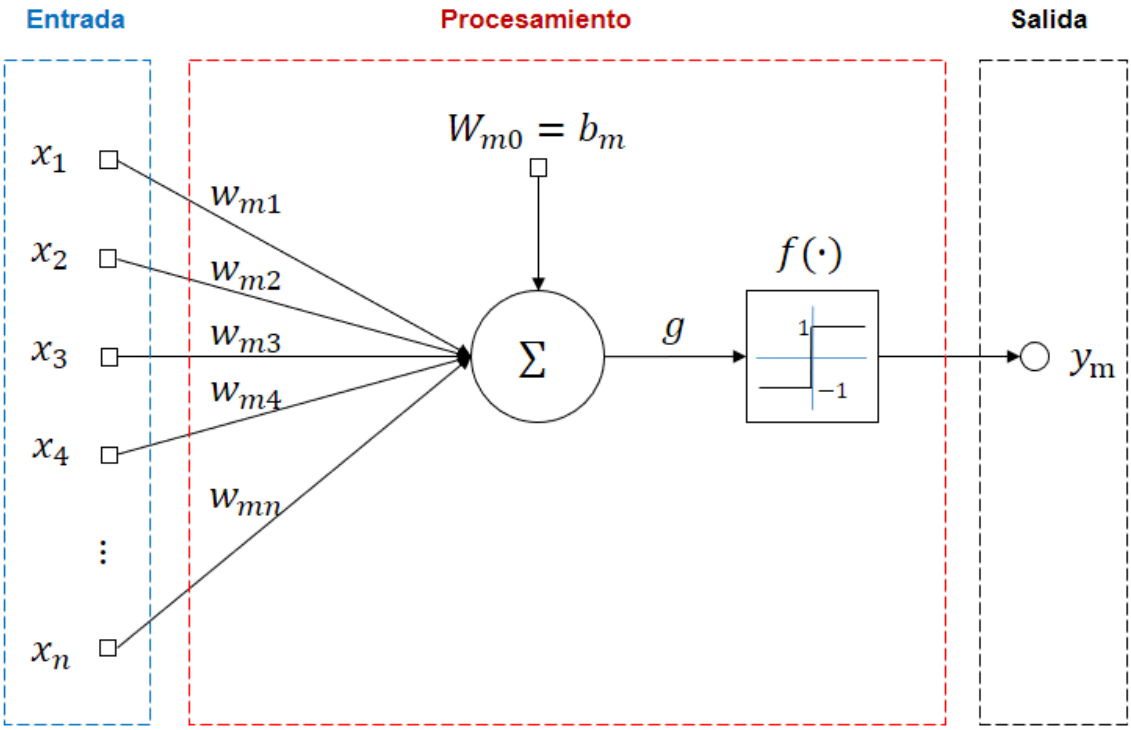
Una neurona es una estructura celular que está dividida en cuatro partes; las dendritas, el cuerpo celular, el axón, y las terminales sinápticas. Las neuronas son la unidades funcionales del cerebro, su textura es delgada y del tamaño de un grano de arroz [27], hay aproximadamente 10 billones de ellas en la corteza humana, con 60 trillones de conexiones o sinapsis [28]. Todos los enlaces construyen una red neuronal biológica que habilita al ser humano de adquirir experiencia de su entorno que lo rodea. La neurona posee un comportamiento afín a un sistema computacional de entrada – salida, análogamente es asociado a una unidad de procesamiento de señales eléctricas.

Las dendritas son ramificaciones que reciben las señales de entrada, son fortalecidas o debilitadas basándose en cómo estas sean usadas (ej. Aprendizaje de nuevos conceptos), y determinan la contribución de las entradas hacia otra neurona, mismas que son transmitidas por medio de las terminales sinápticas de la neurona actual. Dichas entradas son pesadas por las fuerzas de sus respectivas conexiones, todas son sumadas al ingresar al cuerpo celular formando así una nueva señal, la cual se propaga a lo largo del axón hasta ser expulsada por medio de las terminales sinápticas hacia otras neuronas [27]. La sinapsis es la conexión entre neuronas que expone excitación o inhibición, pero no ambos al mismo tiempo, siendo ella en vinculo en la transmisión de información de la red neuronal.

Las Artificial Neural Networks (ANNs) han sido creadas con el fin de emular el comportamiento y la forma de procesar la información de las redes neuronales biológicas. Su desarrollo ha sido posible gracias los estudios del cerebro así como del sistema nervioso central, simulando la actividad eléctrica que ambos sistemas producen [29]. El interés de estudiar las redes neuronales nace del hecho de introducir el modelo de neuronas simplificadas dado por McCulloch y Pitts en 1943, permitiendo establecer una generalidad matemática que describe el funcionamiento de las ANNs [30]. Los elementos clave encontrados en la arquitectura de las ANNs son las neuronas artificiales; ellas a su vez son designadas como nodos, perceptrones, o simplemente neuronas.

La Figura 2.1.1-1 muestra el modelo de una neurona en una ANN actuando como un sistema de entrada – salida, dividido en tres partes; la entrada, el procesamiento, y la salida:

Figura 2.1.1-1: Modelo de una neurona en una ANN



Nombre de la fuente: Elaboración propia

Basado en la ilustración anterior, el conjunto $X \coloneqq \{x_0, x_1, \cdots, x_n / x \in \mathbb{R} \wedge n \in \mathbb{R}\}$ está compuesto de las entradas en la m-ésima neurona de la red neuronal, $W \coloneqq \{w_{m0}, w_{m1}, \cdots, w_{mn} / (m,n) \in \mathbb{Z}^+ \wedge w \in \mathbb{R}\}$ es el conjunto de pesos sinápticos asociados a cada entrada, b_m equivale al bias (para este caso de neurona m), $f(\cdot)$ es la función de activación y y_m es la salida sináptica enviada a la siguiente neurona, misma que puede escribirse como sigue:

$$y_m = f(b_m + \boldsymbol{X}^T \boldsymbol{W}) \tag{2.1}$$

Donde

$$\boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \tag{2.2}$$

Es el vector de entradas que llegan a la neurona y

$$\boldsymbol{W} = \begin{bmatrix} w_{m1} \\ w_{m2} \\ w_{m3} \\ \vdots \\ w_{mn} \end{bmatrix} \tag{2.3}$$

El vector de los pesos sinápticos. b_m es el parámetro directamente responsable del incremento o decremento en la entrada de la función de activación, ello depende de si $b_m \gtrless 0$ [28, p. 42]. v_m se conoce como potencial de activación o campo local inducido y está dado por:

$$v_m = u_m + b_m \tag{2.4}$$

Donde u_m es la salida producida por la combinación lineal, b_m influye significativamente en la magnitud de v_m, entonces se dice que el uso del bias proporciona una *transformación afín* a la salida de la combinación lineal u_m del modelo representado en la Figura 2.1.1-1. Dicha salida está dada mediante la ecuación (2.5):

$$u_m = \boldsymbol{X}^T \boldsymbol{W} \tag{2.5}$$

Por último, es fácil probar que $\boldsymbol{X}^T \boldsymbol{W} = \boldsymbol{W}^T \boldsymbol{X} = u_m$.

2.1.2 Arquitecturas de una red neuronal

Las arquitecturas (topologías) de las ANNs varían dependiendo de la cantidad de variables del vector de entrada, los nodos neuronales, sus conexiones sinápticas, y las salidas de la red. En la Figura 2.1.2-1 pueden apreciarse dos topologías; la arquitectura expuesta en (a) corresponde a una ANN de una sola capa, mientras que la de (b) ejemplifica una red multicapa. Las capas están compuestas por varios nodos neuronales como el de la Figura 2.1.1-1. Las redes que tienen una o más capas en medio de la entrada y salida son como la de la topología ilustrada en la Figura 2.1.2-1 (b), dichas capas son denominadas *capas ocultas*, así, cada neurona perteneciente a ellas se llamará *nodos neuronales ocultos*, *neuronas ocultas*, o *unidades ocultas*. A diferencia de la red de la Figura 2.1.2-1 (a), las

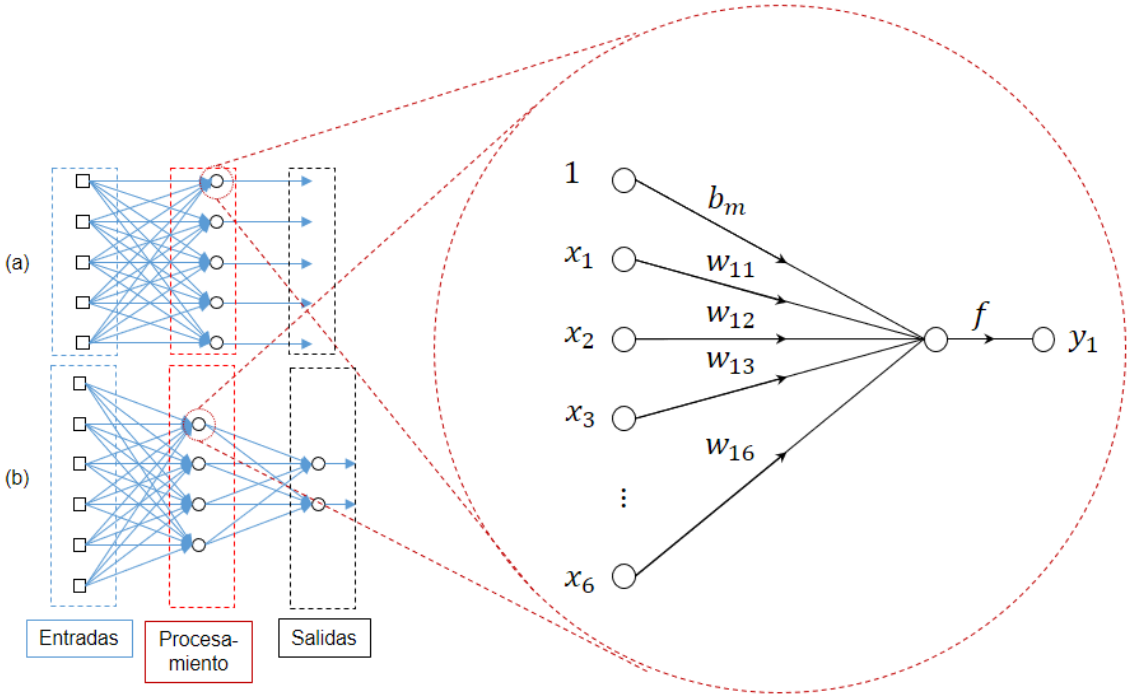
capas ocultas no son un vínculo directo entre la entrada y la salida sino unidades de procesamiento separadas.

La forma en la cual las neuronas están estructuradas está íntimamente ligada al algoritmo de aprendizaje de la ANN [28]. Las capas ocultas propician a un incremento en la robustez de la solución de un problema complejo, como es el caso de la predicción o clasificación de señales o imágenes, entre más capas ocultas mayor es el aprendizaje adquirido en la red a partir de su entrenamiento. Las ANNs con una sola capa oculta crean un hiperplano, una con dos combinan hiperplanos conformando áreas convexas de decisión, tres capas manipulan dichas áreas para constituir otras que contengan regiones cóncavas, de esa forma, la convexidad y concavidad suministran inferencia o abstracción en la región de decisión en función del vector de entrada para producir las salidas deseadas [29, p. 639].

La cantidad de capas y nodos ocultos varían el tiempo en el que la red tarda en ser entrenada. Por ejemplo, una ANN con muchas neuronas y capas ocultas propician largos periodos de entrenamiento comparadas con las que poseen pocos nodos ocultos. Sin embargo esto último, causa una carencia significativa de detectores de características, razón por la cual, la exactitud y el aprendizaje pueden verse afectados [29] y [30]. Asimismo, una red neuronal con m nodos fuente (vector de entrada), h_1 neuronas en la primera capa oculta, h_2 neuronas en la segunda, h_3 neuronas y así sucesivamente hasta h_n, además de q nodos en la capa de salida, es denominada generalmente como una red $m \rightarrow h_1 \longrightarrow h_2 \cdots h_n \longrightarrow q \; \forall (n > 2) \subset \mathbb{N}$ [28].

Las redes neuronales mostradas en la Figura 2.1.2-1 se denominan Feedforward Neural Networks (FNN), puesto que no hay lazos que permitan comparar la salida con la entrada, las Recurent Neural Networks (RNNs) si poseen esta particularidad puesto que los lazos de realimentación involucran elementos de retardo en el tiempo, simbolizados como z^{-1}, derivando en un comportamiento dinámico no lineal, cuando se asume que la red contiene unidades no lineales [28] y [30]. Sin embargo, el estudio detallado de las RNNs está fuera del alcance de este trabajo puesto que ellas son aplicadas a señales modeladas como series temporales para reconocimiento de la voz, clasificación de señales de encefalografía, entre otras, mientras que las CNN si serán abordadas con mayor detalle ya que su arquitectura es similar a la de la Figura 2.1.2-1 (b) y son usadas principalmente en la clasificación de imágenes.

Figura 2.1.2-1: Algunos tipos de arquitecturas de redes neuronales



Nombre de la fuente: Elaboración propia

2.2 El perceptrón de Rosenblat

Desde que McCulloch y Pitts en 1943 introdujeron el concepto de una red neuronal como una maquina computacional [28], que incluso llegó a ser implementada en hardware como pequeñas tarjetas electrónicas [31], naciendo la primera regla de Hebb para el aprendizaje auto-organizado en 1954 [28], así como el modelo de Rosenblat para el perceptrón en 1958 como algoritmos de aprendizaje supervisado [28], han sido los fundamentos teóricos que conforman los cimientos del aprendizaje profundo o Deep Learning (DL). En esta sesión se aborda un modelo llamado *perceptrón de Rosenblat* cuya región de decisión está divida en dos clases, proporcionando una salida binaria, su estructura es expuesta en la Figura 2.1.1-1.

Este perceptrón está basado en el modelo de la neurona no lineal de McCulloch y Pitts [28, p. 48]. El vector de entrada $\boldsymbol{X} \coloneqq [x_1 \quad x_2 \quad \dots \quad x_n]^T$, $\boldsymbol{W} \coloneqq [w_1 \quad w_2 \quad \dots \quad w_n]^T$ son los

pesos sinápticos de la red, $f(\cdot)$ es un limitador o función de umbral conocida como *función de activación* [31, p. 67], y $y \in [-1,1]$. La salida g generada por el nodo suma se escribe como sigue:

$$g(x) = \boldsymbol{X}^T\boldsymbol{W} + b \tag{2.6}$$

La salida y produce dos posibles valores al cruzar por el limitador, esto es:

$$y = \begin{cases} 1 & \text{sí } g(x) > 0 \\ -1 & \text{sí } g(x) < 0 \end{cases} \tag{2.7}$$

Donde y representa un comportamiento equivalente a la función $x(t) = sng(t)$. La región de decisión creada por el hiperplano de la red neuronal del Figura 2.2-1 está dado cuando la función $g(x) = 0$, también conocida como entrada neta de la función o neurona de activación. De este modo el hiperplano es:

$$x_1w_1 + x_2w_2 + x_3w_3 + \cdots + x_nw_n + b = 0 \tag{2.8}$$

Así la heurística del sistema se fundamenta en dos clases; C_1 y C_2, C_1 se asocia a la salida y cuando está equivale a 1, asimismo, si $y = -1$, la salida pertenecerá a la clase C_2. Para dos entradas x_1 y x_2 la región de decisión está dada por el hiperplano de la ecuación (2.9).

$$x_1w_1 + x_2w_2 + b = 0 \tag{2.9}$$

Nótese que $-w_1/w_2$ es la pendiente de la recta definida por la ecuación (2.9) y el punto de corte sobre el eje vertical está dado por la razón $-b/w_2$, esto significa que al entrenar adecuadamente la red neuronal, los pesos sinápticos así como el bias, son ajustados de tal forma que se obtenga el resultado (o decisión) deseado. No obstante, el vector de los pesos siempre esta perpendicularmente con respecto al límite de decisión, si dicho vector apunta hacia arriba se obtiene una clasificación adecuada, en caso contrario serian incorrectas.

2.2.1 Teorema de convergencia del perceptrón

Este teorema establece la convergencia de la salida del perceptrón mostrado en la Figura 2.2-1 a dos clases; C_1 y C_2. Cada clase tiene asociado un valor de decisión para el cual dicho perceptrón ha sido entrenado, por lo que, sus pesos sinápticos han de ajustarse durante el proceso de entrenamiento para obtener la salida deseada, el bias ofrece un grado de libertad adicional en la región de decisión. En ese sentido el teorema tiene dos postulados [28, p. 51]:

Primer postulado: si el $k-ésimo$ elemento del conjunto de entrenamiento $\boldsymbol{X}(k)$ es clasificado correctamente por el vector de pesos $\boldsymbol{W}(k)$ calculado en la $k-ésima$ iteración no hay corrección en el vector de pesos de acuerdo con la regla:

$$\begin{aligned}\boldsymbol{W}(k+1) &= \boldsymbol{W}(k) \Leftrightarrow \boldsymbol{W}^T(k)\boldsymbol{X}(k) > 0 \wedge \boldsymbol{X}(k) \in C_1 \\ \boldsymbol{W}(k+1) &= \boldsymbol{W}(k) \Leftrightarrow \boldsymbol{W}^T(k)\boldsymbol{X}(k) \leq 0 \wedge \boldsymbol{X}(k) \in C_2\end{aligned} \tag{2.10}$$

Segundo postulado: en contradicción al primer postulado los pesos del perceptrón son actualizados de acuerdo a la regla:

$$\begin{aligned}\boldsymbol{W}(k+1) &= \boldsymbol{W}(k) - \alpha(k)\boldsymbol{X}(k) \Leftrightarrow \boldsymbol{W}^T(k)\boldsymbol{X}(k) > 0 \wedge \boldsymbol{X}(k) \in C_2 \\ \boldsymbol{W}(k+1) &= \boldsymbol{W}(k) + \alpha(k)\boldsymbol{X}(k) \Leftrightarrow \boldsymbol{W}^T(k)\boldsymbol{X}(k) \leq 0 \wedge \boldsymbol{X}(k) \in C_1\end{aligned} \tag{2.11}$$

Donde $(\{\boldsymbol{X}_1(k)\} \in H_1) \cup (\{\boldsymbol{X}_2(k)\} \in H_2)$ son los elementos de los datos de entrenamiento particionados de los subconjuntos H_1 y H_2, donde $C_1 \subseteq H_1 \wedge C_2 \subseteq H_2$, H esta compuesto por la totalidad de los datos de entrenamiento, es decir, $H_1 \cup H_2 = H$, $\alpha(k)$ es la rata de entrenamiento que controla los ajustes aplicados al vector de pesos de la iteración k. La demostración a este teorema no será mostrada en este capítulo pero puede referirse a [28, p. 52] para mayor información.

2.2.2 Algoritmo del perceptrón por lotes

Este algoritmo está definido mediante la siguiente ecuación [28, p. 65]:

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + \alpha(k) \cdot \sum_{\boldsymbol{X}(k)\in\widehat{\boldsymbol{X}}} \boldsymbol{X}(k) y_d(k) \tag{2.12}$$

Donde $y_d(k)$ es la salida deseada para la región de decisión y $\widehat{\boldsymbol{X}}$ es el conjunto de entrada para entrenar el perceptrón, los pesos son ajustados de acuerdo la rata de aprendizaje y el gradiente de la llamada *función de costo del perceptrón*. Dicha función es descrita como sigue:

$$\boldsymbol{J}(\boldsymbol{W}) = \sum_{\boldsymbol{X}(k)\in\widehat{\boldsymbol{X}}} -\boldsymbol{W}^T \boldsymbol{X}(k) y_d(k) \tag{2.13}$$

Así el gradiente de la ecuación (2.13) está dado por:

$$\boldsymbol{\nabla J}(\boldsymbol{W}) = -\sum_{\boldsymbol{X}(k)\in\widehat{\boldsymbol{X}}} \boldsymbol{X}(k) y_d(k) \tag{2.14}$$

Donde $\boldsymbol{\nabla}$ es el operador gradiente dado por el siguiente vector de derivadas parciales:

$$\boldsymbol{\nabla} = \begin{bmatrix} \frac{\partial}{\partial w_1} \\ \frac{\partial}{\partial w_2} \\ \vdots \\ \frac{\partial}{\partial w_n} \end{bmatrix} \tag{2.15}$$

2.2.3 Funciones de activación

Las funciones de activación establecen la región de decisión (como se muestran en la Figura 2.2-2) del hiperplano generado en la salida del perceptrón, su propósito es que una ANN aprenda la estructura de las variables de entrada, ya sean lineales o no lineales [32], dichas funciones también son conocidas como neuronas de activación siendo tres de ellas las más usadas en la construcción de ANNs [27], definidas como sigue:

Función Sigmoide: la neurona de activación sigmoidal y su derivada en el dominio $\boldsymbol{W}^T\boldsymbol{X}$ están dadas por las siguientes ecuaciones:

$$f_{sigmoide}(\boldsymbol{W}^T\boldsymbol{X}) = \frac{e^b}{e^b+e^{-\boldsymbol{W}^T\boldsymbol{X}}} \tag{2.16}$$

Su derivada es

$$f'_{sigmoide}(\boldsymbol{W}^T\boldsymbol{X}) = f_{sigmoide}(\boldsymbol{W}^T\boldsymbol{X}) \cdot \left[1 - f_{sigmoide}(\boldsymbol{W}^T\boldsymbol{X})\right] \tag{2.17}$$

La función $f_{sigmoide}(\boldsymbol{W}^T\boldsymbol{X}): \mathbb{R} \to [0,1]$ es plana y de clase C^1 en $\mathbb{R}$, ella posee dos problemas al ser implementada en la construcción de redes neuronales: primero, no se aproxima a la función identidad cerca del origen, y segundo, su derivada cerca del origen no está encerrada en 1. Al ser una función de aplanamiento, su saturación ocurre cuando $\boldsymbol{W}^T\boldsymbol{X}$ incrementa considerablemente, su gradiente es muy pequeño si $\boldsymbol{W}^T\boldsymbol{X}$ no está encerrado en el origen. En una red con muchas capas conformadas con neuronas de activación sigmoildales los cambios de los pesos son muy pequeños o despreciables, lo cual causa que la red se estanque y no aprenda, por ello dicha función no es usada en redes DL [32, p. 72].

Función Tangente Hiperbólica: también es una función analítica de clase C^1 y $f_{tanh}(\boldsymbol{W}^T\boldsymbol{X}): \mathbb{R} \to [-1,1]$ conocida como una versión escalada de la neurona sigmoidal, ella se define como sigue:

$$f_{tanh}(\boldsymbol{W}^T\boldsymbol{X}) = 2 \cdot \left(\frac{e^{2b}}{e^{2b}+e^{-2\boldsymbol{W}^T\boldsymbol{X}}}\right) - 1 \tag{2.18}$$

Su primera derivada es

$$f_{tanh}(\boldsymbol{W}^T\boldsymbol{X}) = 1 - [f_{tanh}(\boldsymbol{W}^T\boldsymbol{X})]^2 \tag{2.19}$$

Si bien, esta neurona se aproxima a la función lineal a medida de que $\boldsymbol{W}^T\boldsymbol{X} \to 0$ tiene el mismo problema de saturación de la sigmoide. Ambas neuronas se ven envueltas en las dificultades causadas por el desvanecimiento del gradiente cuando la ANN posee muchas capas. Básicamente, se atribuyen a la estimación del error y por lo tanto al cálculo del gradiente de perdida, aun así, predomina el uso de la función tangente hiperbólica con respecto a la sigmoide por el hecho de su proximidad con la identidad.

Función Rectifier Linear Unit (ReLU): es una neurona computacionalmente muy eficiente, definida como sigue:

$$f_{ReLU}(\boldsymbol{W}^T\boldsymbol{X}+b)=\max(0,\boldsymbol{W}^T\boldsymbol{X}+b) \tag{2.20}$$

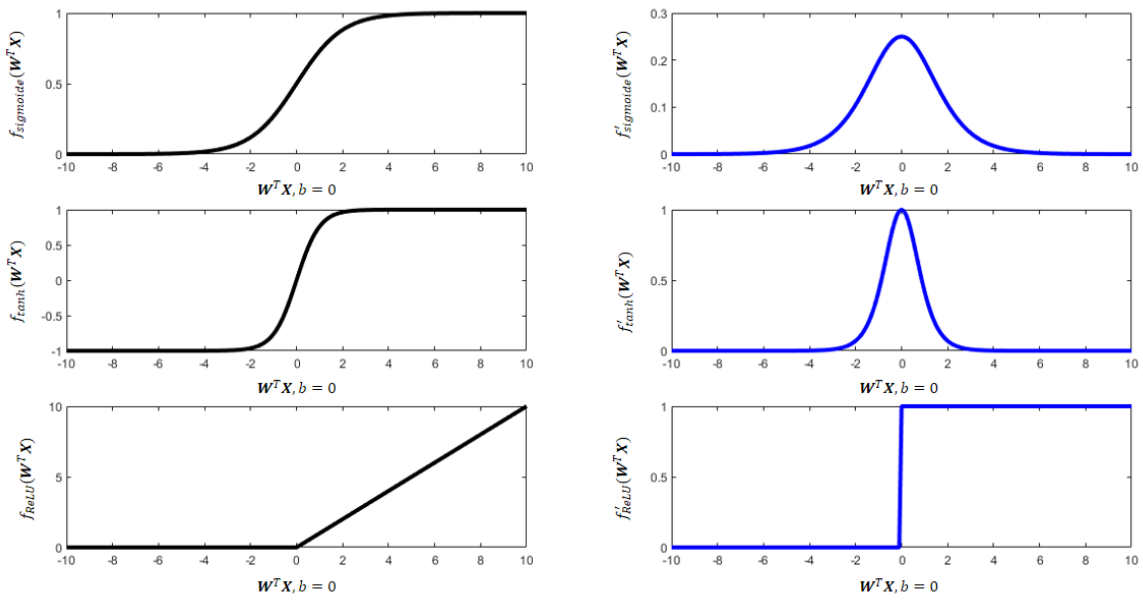
Su primera derivada está dada por

$$f'_{ReLU}(\boldsymbol{W}^T\boldsymbol{X}+b)=\begin{cases}0,\boldsymbol{W}^T\boldsymbol{X}+b<0\\1,\boldsymbol{W}^T\boldsymbol{X}+b\geq 0\end{cases} \tag{2.21}$$

Esta función evita el desvanecimiento del gradiente en redes de muchas capas lo cual la hace apropiada para su implementación en redes neuronales profundas. Una red profunda es un tipo de red FNN con muchas capas en su arquitectura, las redes con muy pocas capas en su arquitectura pertenecen a las redes neuronales poco profundas. Una particularidad de esta función de activación es que puede ocasionar muerte de neuronas durante el entrenamiento. Una neurona muerta siempre retorna un cero para cada muestra de la base de datos. Esto puede suceder porque los pesos de una neurona muerta habrán de ser ajustados tal que $x_n w_n < 0$, donde el subíndice $n = \{1,2,\dots N\}$ determina la neurona de la red.

Su ventaja es que la salida de una capa puede tener enteros los cuales son siempre cero, por lo cual pueden despreciarse para hacer computacionalmente más eficiente la red. Su desventaja es que, la muerte de neuronas puede afectar la exactitud de la ANN [32, p. 75]. La Figura 2.2.3-1 muestra las funciones de activación anteriormente comentadas junto con su primera derivada, en ella puede apreciarse el comportamiento de linealidad en cada una de ellas, así como la dinámica de cada uno de sus gradientes.

Figura 2.2.3-1: Funciones de activación y sus derivadas.



Nombre de la fuente: Elaboración propia

2.2.4 El perceptrón multicapa

El Multi-Layer Perceptrón (MLP) es una extensión del perceptrón de Rosenblat. Básicamente, ha surgido para efectuar tareas de clasificación de comportamiento no lineal. El MLP conforma una red multicapa similar a la de la Figura 2.1.2-1(b), compuesta por nodos de entrada, capas ocultas y los nodos de salida. Los nodos de entrada reciben señales cuyo comportamiento es de una función de señal debido a dos razones; uno, puede usarse como función en la salida de la red, y dos, en cada neurona en la cual dicha función pasa, la señal es calculada como una función de las entradas asociando los pesos sinápticos de esa neurona [28].

Cada neurona de salida u oculta de una red MLP es diseñada para ejecutar dos cálculos: el primero corresponde al cálculo de la función de señal que aparece en la salida de cada neurona, la cual es expresada como una combinación no lineal de las señales de entrada y sus pesos sinápticos, y el segundo obedece a estimar el vector gradiente que es necesario para el paso inverso a través de la red. Así, las neuronas ocultas operan como detectores de características jugando un rol crucial en la funcionalidad de un MLP [28]. Por

último, una señal de error yace en los nodos de salida y se propaga hacia atrás (capa por capa) a lo largo de toda la red. Se denomina señal de error porque su cálculo desde cada neurona involucra una función dependiente del error de una forma u otra [28].

2.3 Redes neuronales convolucionales

Las CNN (o ConvNets) nacen a partir de la motivación de estudiar el comportamiento de la corteza visual del ser humano, aquí la extracción de características de una imagen es hecha jerárquicamente, desde el bajo, medio hasta el alto nivel. La primera ConvNet fue creada por Yan LeCun en 1998 y se llamó LetNet, pero fue en el año 2012 donde ellas mostraron su verdadera potencia ante la comunidad científica, mostrando un alto desempeño en tareas de reconocimiento de imágenes, sus resultados fueron mostrados en la competencia *Imagenet* dada en el mismo año [33, p. 59]. Esta propiedad ha propiciado su uso en diversos campos del conocimiento para clasificar imágenes, por ejemplo en la detección temprana de enfermedades en cultivos [34] y [35]. Su arquitectura está dividida en tres secciones que son; pre –procesamiento de imágenes, extracción de características y clasificación [36].

2.3.1 Pre – procesamiento

Inicialmente debe efectuarse una etapa de pre – procesamiento la cual incluye mecanismos de segmentación que configuran las imágenes de entrada a escala de grises. Por esta razón, (…) “hay un número de operaciones que pueden utilizarse en orden de hacer del entrenamiento de una CNN ligeramente más fácil. Una de ellas, consiste en usar la técnica que está apoyada en la librería de TensorFlow para Python que consiste en una aproximación por blanqueamiento de imagen. La idea básica consiste en un centro cero para cada pixel en una imagen, restando la media y normalizando la varianza a 1” (…) [37, p. 110]. Asimismo, la selección de la resolución adecuada, la transformación de la escala de grises y su segmentación en las componentes Red Green and Blue (RGB) son otras operaciones aplicadas como técnicas de pre – procesamiento para el Digital Image Processing (DIP) [35] y [19].

2.3.2 Capa convolucional

Comprende efectuar la operación de convolución de la imagen de entrada con matrices de 3 x 3 o de 5 x 5 denominadas kernels [37, p. 99], [19] y [38] para la configuración de filtros que apoyan a la extracción de características de la matriz de entrada viniente [38]. Los filtros con los kernels más pequeños logran una alta potencia representacional mientras que ocurre el más pequeño número de parámetros, a su vez, usar un paso de 1 en la operación de convolución construye los llamados mapas de características [38], así como un relleno de ceros o zero padding estructura a la matriz de salida con las mismas dimensiones que la de entrada [37, p. 104]. Formalmente la operación de convolución puede ser escrita de la forma [32] y [39, p. 167]:

$$y(n,m) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)x(n+s,m+t) \tag{2.22}$$

Donde $(a \wedge b) \in \mathbb{Z}^+$, (s,t) son los ejes de traslación en el espacio vertical y horizontal respectivamente,

$$x(n,m) = \begin{bmatrix} x(0,0) & \cdots & x(0,M-1) \\ \vdots & \ddots & \vdots \\ x(N-1,0) & \cdots & x(N-1,M-1) \end{bmatrix} \tag{2.23}$$

Es la imagen de entrada a la CNN representada matemáticamente como el arreglo matricial de la Ecuación (2.23), $w(s,t)$ es el filtro o kernel para obtener el mapa de características de la imagen $x(n,m)$ y está definida como sigue:

$$w(s,t) = \begin{bmatrix} w(0,0) & \cdots & w(0,T-1) \\ \vdots & \ddots & \vdots \\ w(S-1,0) & \cdots & w(S-1,T-1) \end{bmatrix} \tag{2.24}$$

En efecto, $x(n,m)$ puede representar una imagen binaria o en escala de grises, ella así como $w(s,t)$ son espacios vectoriales en $\mathbb{R}^2$, es decir que, $\boldsymbol{X} \in \mathbb{R}^{n \times m} \wedge \boldsymbol{W} \in \mathbb{R}^{s \times t}$. Para el caso de una imagen a color o multicanal cuyas componentes base son el color Rojo, Verde y Azul (RGB por sus siglas en ingles) $\boldsymbol{X} \in \mathbb{R}^{n \times m \times 3}$, por lo tanto el kernel $w(s,t)$ es una matriz $\boldsymbol{W} \in \mathbb{R}^{s \times t \times 3}$, donde $n < s \wedge m < t$. La convolución de ambas matrices simbolizada como $\boldsymbol{X} * \boldsymbol{W}$ resulta en una matriz uni – canal $\boldsymbol{Y} \in \mathbb{R}^{(n-s+1) \times (m-t+1) \times 1}$, ello traduce la

facilidad de implementar varias capas de convolución [32, p. 89]. La representación del filtro convolucional puede escribirse en forma lineal por medio de la ecuación (2.6) donde b puede ser un peso de tendencia o umbral [39] y [33, p. 168].

2.3.3 Capa de activación

La transformación no lineal puede aplicarse a la salida de la convolución en una CNN, el objetivo es clasificar las características dentro de las capas ocultas [19, p. 260]. La función de Rectified Linear Units (ReLU) es frecuentemente usada para la activación y ejecuta una simple operación de umbral, donde cualquiera valor de entrada más pequeño que cero es fijado en cero. Asimismo, la función Sigmoide así como la Tangente Hiperbólica también pueden ser usadas para esta capa, aunque vale aclarar que posee las limitantes anteriormente mencionadas [38].

2.3.4 Capa de agrupamiento

El objetivo de usar esta capa es el de reducir las dimensiones de los mapas de características, los enfoques más usados son agrupación promedio y agrupación máxima o "*max pooling*" [10], esta última extrae los parches de los mapas de características de entrada descartando otros valores [38].

2.3.5 Capa de conexión completa

Para la clasificación de las imágenes se aplicaran las capas completamente conectadas o FCL (por sus siglas en inglés) cuyas salidas están conformadas por el número de clases de clasificación [38, p. 616], para el caso de esta propuesta será la clasificación de imágenes de hojas con y sin tizón tardío, la cual se obtendrá a través del ajuste de las capas de clasificación de la CNN implementada.

2.4 Entrenamiento de redes neuronales

El entrenamiento de las redes neuronales consiste en *sintonizar* los pesos y el bias de todas las neuronas de la red con el fin de que ella aprenda, dicha actualización se basa en el cálculo del gradiente descendente. Existen varias técnicas para optimizar el proceso de entrenar una red y así buscar que el error sea el mínimo posible, con base en la obra [33] del profesor PhD. Jesús A. López S van a describirse los algoritmos de entrenamiento de los siguientes apartes:

2.4.1 Gradiente descendente para redes poco profundas

Esta técnica consiste en buscar el error mínimo posible en la función de perdida de modo que puedan ajustarse los pesos sinápticos de la red, en función de producir la menor cantidad de error a la salida. El gradiente descendente que permite actualizar los pesos se escribe como sigue:

$$\Delta w_i(t) = -\alpha \frac{\partial L}{\partial w_i(t)} \tag{2.25}$$

Donde $\Delta w_i(t)$ es la variación del peso sináptico de la i-ésima neurona, α es el factor de velocidad de aprendizaje, y L es la función de perdida (o costo) descrita por la ecuación del error cuadrado medio (MSE, por su siglas en inglés) dada por:

$$L = \frac{1}{P}\sum_{p=1}^{P}\left(y_{dp} - y_p\right)^2 \tag{2.26}$$

Donde P es el p-ésimo patrón de entrada del conjunto de entrenamiento, y_{dp} es la salida deseada para el p-ésimo patrón siendo y_p es la salida de la neurona. Por conveniencia, se hace la sustitución de k por t para mayor comodidad, de tal manera que la ecuación de actualización de pesos usando el gradiente descendente es:

$$\Delta w_i(t+1) = w_i(t) + \frac{2}{P}\alpha e_p x_i \tag{2.27}$$

Donde $e_p := y_{dp} - y_p$. Este algoritmo solo actualiza los pesos sinápticos de redes similares a la mostrada en la Figura 2.1.2-1. Para redes que posean más de dos capas ocultas se evidencia una dificultad al momento de calcular el error, puesto que no es posible tener un valor deseada en la salida las neuronas de cada capa oculta. Para explicar el origen de este problema supóngase que se desea entrenar una red con muchas capas ocultas, la cual tiene el vector de entrada de la base en entrenamiento la Ecuación (2.28):

$$\boldsymbol{X}_p = [x_{p1} \quad \cdots \quad x_{pN}]^T \tag{2.28}$$

La salida neta de la i-ésima neurona de la j-ésima capa oculta está dada por

$$g_{pj}^O = \sum_{i=1}^N x_{pi} w_{ji}^O + b_j^O \tag{2.29}$$

Donde g_{pj}^O es la salida de la combinación lineal de la i-ésima neurona, w_{ji}^O es la conexión sináptica entre la neurona i-ésima de la entrada y la j-ésima capa oculta, b_j^O es el bias asignado por la neurona de esa capa oculta. La salida de la neurona hacia la siguiente capa es:

$$y_{pj}^O = f_j^O\left(g_{pj}^O\right) \tag{2.30}$$

Donde $f_j^O(\cdot)$ representa la función de activación para la neurona de la j-ésima capa oculta. Análogamente, la entrada de la k-ésima neurona de la capa de salida es:

$$g_{pk}^S = \sum_{j=1}^O y_{pj}^O w_{kj}^S + b_k^S \tag{2.31}$$

La salida es

$$y_{pk} = f_k^S\left(g_{pk}^S\right) \tag{2.32}$$

Así, al usar el gradiente descendente la actualización de los pesos asociados a la capa de salida es:

$$w_{kj}^{S}(t+1) = w_{kj}^{S}(t) + \frac{2}{P}\alpha\delta_{k}^{S}y_{pj}^{O} \tag{2.33}$$

El error visto a la entrada de la neurona de la capa de salida se define como $\delta_k^S = e_{pk}f_k'^S(g_{pk}^S)$, $e_{pk} = y_{dpk} - y_{pk}$ es el error de salida producido por la neurona de la k-ésima capa (salida), y $f_k'^S$ es la primera derivada de la función de activación. Del mismo modo, la actualización del bias está dada como sigue:

$$b_{k}^{S}(t+1) = b_{k}^{S}(t) + \frac{2}{P}\alpha\delta_{k}^{S} \tag{2.34}$$

El error visto a la entrada de esta neurona es δ_k^S. La actualización de los pesos en la capa oculta está dada por la ecuación (2.35), es decir:

$$w_{ji}^{O}(t+1) = w_{ji}^{O}(t) + \frac{2}{P}\alpha\delta_{j}^{O}x_{i} \tag{2.35}$$

Donde $\delta_j^O = e_{pj}^O f_j'^O(g_{pj}^O)$ es el error visto en la entrada de la neurona de la j-ésima capa oculta, es evidente que este error no puede estimarse ya que no se conoce el valor deseado y_{dpj}^O por lo cual se ha presentado la dificultad antes mencionada. Afortunadamente, a través de los años se ha dado alcance a este problema a través de un método llamado Backpropagation o propagación inversa, el cual es explicado a continuación.

2.4.2 Propagación inversa (*Backpropagation*)

Este método consiste en estimar el error de una neurona propagando los errores de las neuronas de la capa siguiente en sentido inverso. Matemáticamente dicho error se define de la siguiente forma:

$$e_{pj}^{O} = \sum_{k=1}^{M}\delta_{k}^{S}\,w_{kj}^{S} \tag{2.36}$$

Usando la Ecuación (2.36) en la ecuación (2.35) la actualización de los pesos sinápticos se escribe como sigue:

$$w_{ji}(t+1) = w_{ji}^{O}(t) + \frac{2}{M}\alpha\left(\sum_{k=1}^{M}\delta_k^S w_{kj}^S\right)\cdot f_j'^{O}\left(g_{pj}^{O}\right)\cdot x_i \tag{2.37}$$

Del mismo modo, la actualización del bias en la j-ésima capa oculta está dada por

$$b_j^{O}(t+1) = b_j^{O}(t) + \frac{2}{M}\alpha\left(\sum_{k=1}^{M}\delta_k^S w_{kj}^S\right)\cdot f_j'^{O}\left(g_{pj}^{O}\right) \tag{2.38}$$

Las ecuaciones (2.37) y (2.38) permiten la correcta actualización de pesos en redes poco profundas. No obstante, hay dos consideraciones a tenerse en cuenta para el entrenamiento de redes DL; la primera, es el uso de una función de activación que sea derivable y la segunda, se atribuye al empleo de una adecuada función de costo, ambas deben evitar al máximo el problema de desvanecimiento del gradiente.

2.4.3 Cross Entropy Error (CEE)

La función MSE contribuye al problema anteriormente mencionado en redes de aprendizaje profundo, por lo cual, se prefiere usar el error de entropía cruzada [33, p. 125].

$$CEE = -\frac{1}{PM}\sum_{p=1}^{P}\sum_{k=1}^{M} y_{dpk}\ln\left|y_{pk}\right| + \left(1-y_{dpk}\right)\cdot\ln\left|\left(1-y_{pk}\right)\right| \tag{2.39}$$

Una de las razones para el uso de la ecuación (2.39) sobre el MSE es evitar el problema del desvanecimiento gradiente descendente, a pesar del uso de funciones sigmoidales. De este modo, existe una proporcionalidad del peso sináptico hacia el error que tiene la neurona en su salida proporcionando una mayor dinámica en la función de actualización de los pesos mejorando el comportamiento de los algoritmos de entrenamiento [33].

2.4.4 Gradiente descendente estocástico con *momentum*

Ha surgido con el fin de atenuar las oscilaciones generadas por el gradiente descendente eligiendo un punto de partida aleatoriamente con el fin de obtener una mayor aproximación al valor mínimo posible, puesto dicho comportamiento evita la convergencia a este valor. La Ecuación (2.40) expresa su comportamiento:

$$\Delta w(t) = \beta \Delta w(t-1) - \alpha \frac{\partial L}{\partial w(t)} \tag{2.40}$$

El parámetro $\beta \in [0,1]$ aumenta la velocidad de convergencia el error más bajo en el entrenamiento de la red, produciendo un efecto de frenado en las oscilaciones presentadas para llegar a este valor.

2.4.5 Gradiente descendente Adam

Es uno de los algoritmos más usados en el entrenamiento de redes neuronales de aprendizaje profundo. Su nombre se deriva de la abreviatura Adaptive Moments, el primer termino de momento acumula el gradiente y el segundo almacena ese valor al cuadrado. Debe calcularse una corrección de desviación o bias a cada *momentum* con el fin de actualizar los pesos. Los dos momentos se fijan en cero, α_0 corresponde a la razón del aprendizaje global, ε es una constante pequeña usada para la estabilidad numérica, $\rho_1 \wedge \rho_2$ se atribuyen al decaimiento para los momentos $M_1(t) \wedge M_2(t)$. Dichos momentos son representados mediante la siguiente ecuación:

$$\begin{cases} M_1(t) = \rho_1 M_1(t-1) + (1-\rho_1)\left(\frac{\partial L}{\partial w_i(t)}\right) \\ M_2(t) = \rho_2 M_2(t-1) + (1-\rho_2)\left(\frac{\partial L}{\partial w_i(t)}\right)^2 \end{cases} \tag{2.41}$$

Las correcciones de cada momento $CM_1(t)$ y $CM_2(t)$, así como el cambio dinámico de los pesos sinápticos están dadas mediante el sistema de ecuaciones (2.42):

$$\begin{cases} CM_1(t) = {M_1(t)}\Big/{(1-\rho_1)} \\ CM_2(t) = {M_2(t)}\Big/{(1-\rho_2)} \\ \Delta w_i(t) = -\left({\alpha_0 CM_1(t)}\Big/{\varepsilon + \sqrt{CM_2(t)}}\right) \end{cases} \tag{2.42}$$

3. Agrupamiento de las bases de datos

Este capítulo muestra las bases de datos usadas para este proyecto que son la de *PlantVillage* y una elaborada por el autor, estructurando su organización. También, se sugieren algunas técnicas de pre – procesamiento para evaluar y atenuar artificios hallados en las imágenes de dichas bases de datos.

3.1 Base de datos PlantVillage

Para la construcción de la herramienta computacional se ha usado la base de datos "PlantVillage" disponible en el sitio web www.plantvillage.org, contiene 152 imágenes concernientes a fotografías a color de hojas sanas de siembra de papa y 1000 hojas enfermas de tizón tardío. Dicha base alberga en su totalidad 54.303 imágenes divididas en 38 categorías por especies y enfermedades, ellas provienen de diversas investigaciones llevadas a cabo por universidades de Estados Unidos, como la Universidad del estado de Penn, la Universidad Estatal de Florida y la Universidad de Cornell, las fotografías fueron capturadas usando la metodología mencionada en [40, p. 7].

Las especificaciones técnicas de cada imagen son las siguientes:

- Dimensión de cada imagen 256 x 256 pixeles.
- Resolución horizontal y vertical de 96 x 96 puntos por pulgada (ppp).
- Profundidad de 24 bits.
- El formato del archivo es en extensión ".jpg"

En [9] y [41] se ha usado dicha base de datos con una distribución del 80% de las imágenes para la fase de entrenamiento y el 20% restante como datos de prueba, con este porcentaje

se estima verificar la exactitud de clasificación en la red neuronal. Por ejemplo; [42] ha tomado 791 de 1000 imágenes para la fase de entrenamiento y 209 para la fase de prueba en el diagnóstico de hojas con tizón tardío, 122 y 30 imágenes de 152 serán usadas para entrenamiento y prueba de hojas sanas, mientras que [43] usa una relación 6:6:2 para entrenamiento, validación y prueba.

3.2 Base de datos propia

Adicionalmente, este trabajo cuenta con una base de datos propia construida a partir de imágenes de hojas sanas y afectadas con Tizón Tardío según apreciaciones dadas por los agricultores, cada una de ellas fueron extraídas de los cultivos ubicados en los corregimientos de Mochuelo Alto y Pasquilla ubicados al sur de Bogotá en la localidad 19 llamada Ciudad Bolívar. Gracias a este trabajo fue posible obtener 166 fotografías de hojas de plantas de papa Criolla y Capiro.

3.3 Organización de las bases de datos

La cantidad de elementos recolectados en este trabajo que conforman la base de datos es de 3622 imágenes, 585 de ellas atribuidas a folios sanos y 3037 a hojas enfermas. Sus especificaciones técnicas se encuentran en la Tabla 3-1:

Tabla 3-1: Construcción de la base de datos del proyecto.

Fuente	Tipo de imagen	Hojas sanas	Hojas enfermas	Dimensión en pixeles	Resolución (ppp)	Formato
PlantVillage	RGB	152	1000	256 x 256	96 x 96	.jpg
	Escala de grises	152	1000	256 x 256	96 x 96	.jpg
	Segmentadas	152	1000	256 x 256	96 x 96	.jpg
Propia	RGB	11	21	3648 x 2736	180 x 180	.jpg
		112	5	2248 x 3264	72 x 72	.jpg
		6	11	3120 x 4160	72 x 72	.jpg

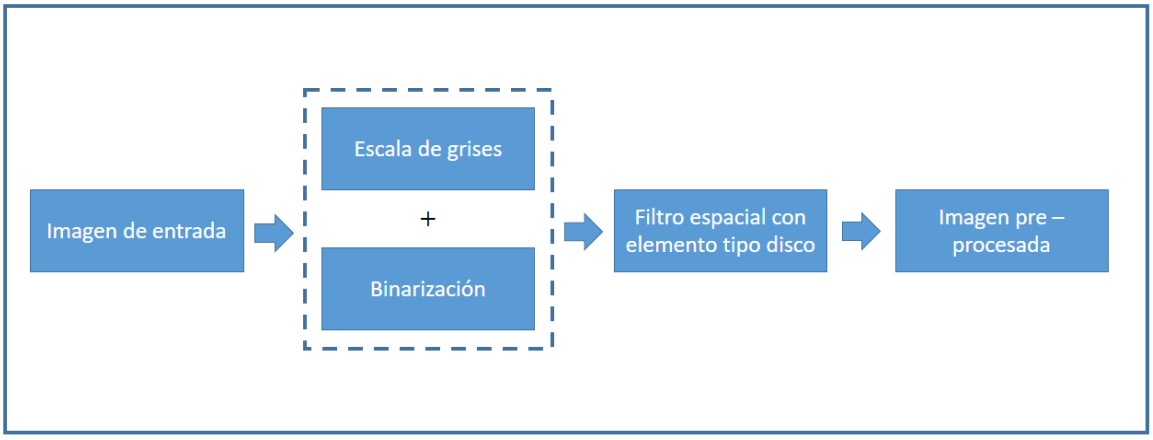
Para este trabajo, la base de datos PlantVillage será dividida en un 80% como señales de entrada en la fase de entrenamiento y el 20% restante apoyaran la validación de la red.

Sin embargo, los registros fotográficos tomados por el autor tienen la finalidad de comprobar la clasificación de imágenes asociadas a cultivos colombianos.

3.4 Etapa de pre – procesamiento

Comúnmente, es recomendable adaptar todas las imágenes a una etapa de pre – procesamiento, permitiendo extraer ciertas características que aumentan la calidad del entrenamiento en la red neuronal [43]. Sin embargo, en este proyecto se estudia esta etapa con el fin de encontrar un filtro que permita atenuar las componentes de ruido de las imágenes de entrada, mismo que será implantado en la herramienta computacional, brindando al usuario la opción de limpiar las imágenes cargadas antes de iniciar con su clasificación. También se incluye el proceso de binarización, pensando en que el usuario pueda apreciar los artefactos que contiene cada imagen cargada a dicha herramienta. La Figura 3.4-1 muestra el ciclo de procesamiento propuesto por el autor:

Figura 3.4-1: Etapa de pre – procesamiento propuesta



Nombre de la fuente: Elaboración propia

3.4.1 Imagen de entrada

La imagen de entrada ha sido modelada como una matriz $\boldsymbol{X} \in \mathbb{R}^{256\times256\times3}$ gracias a la instrucción *imread()* de Matlab, la morfología de dicha imagen suele mostrarse mediante la instrucción *imshow()*, dado que esta interpreta a $\boldsymbol{X}$ como una imagen a color.

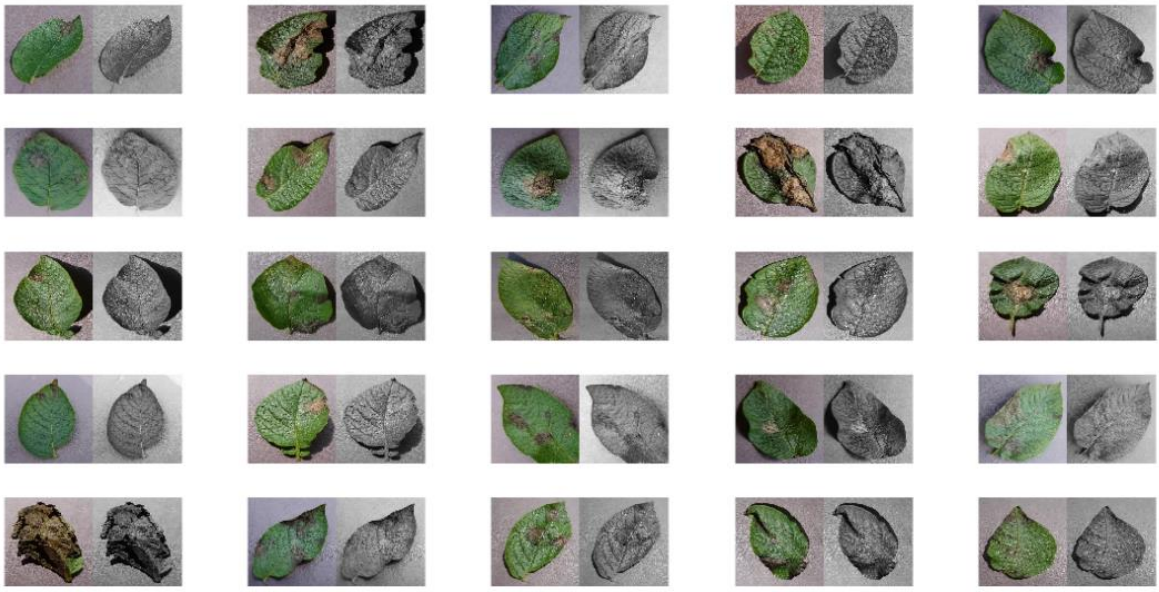
3.4.2 Escala de grises

El empleo de esta conversión es atribuida como el nivel de luminancia de la imagen [44, p. 15] y [45], es posible estimarla a través de la función de intensidad [46] descrita como sigue:

$$\mathcal{G}(x,y,z) = 0.299R(x,y,z) + 0.587G(x,y,z) + 0.114B(x,y,z) \quad (3.1)$$

donde $\mathcal{G}$ simboliza la transformación a escala de grises de una imagen a color dada, $R, G \wedge B \in \mathbb{R}^{256\times256\times3}$ son las componentes de color rojo, verde y azul, por sus siglas en inglés, en el espacio tridimensional, así, los coeficientes que acompañan a cada una de las componentes son una combinación de pesos. La luminancia es el algoritmo estándar usado por software de procesamiento digital de imágenes como es el caso de Matlab, para transformar una imagen $\boldsymbol{X} \in \mathbb{R}^{n\times m\times3}$ a escala de grises, en aplicaciones de visión artificial [45], ella puede emplearse mediante el comando *rgb2gray()*. La Figura 3.4.2-1 bosqueja dicha transformación ante 25 imágenes de hojas de tizón tardío definidas como entradas que van hacia el segundo bloque de la Figura 3.4-1:

Figura 3.4.2-1: Transformación a escala de grises de 25 imágenes de entrada



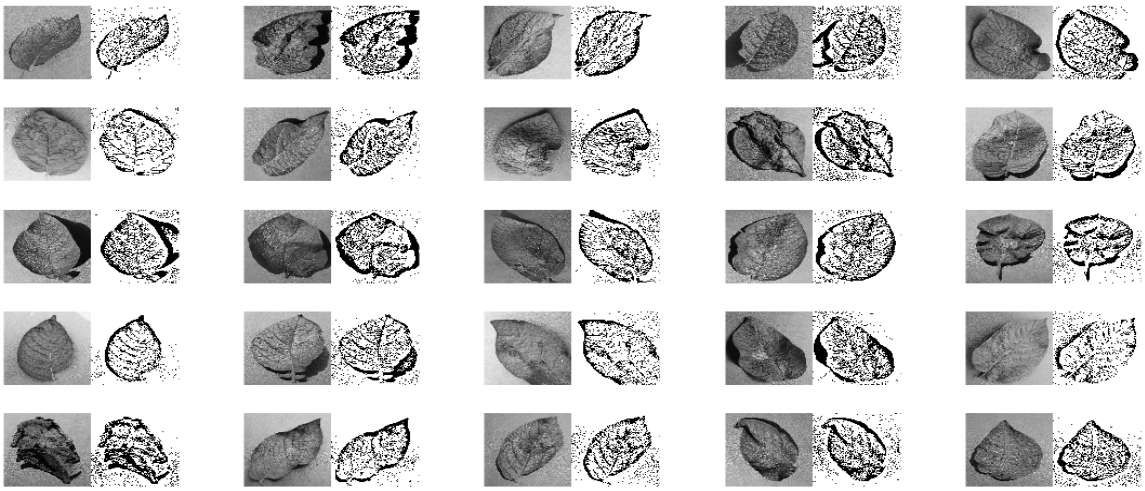
Nombre de la fuente: Elaboración propia

Nótese qué al costado izquierdo de cada imagen de la figura anterior, se encuentran las fotografías de hojas de tizón tardío y al lado derecho su respectiva trasformación a escala de grises, dicha escala es necesaria para proceder con la binarización de las imágenes del lado izquierdo, misma que será implementada en la herramienta computacional para apreciar los artefactos de cada imagen de entrada.

3.4.3 Binarización

Corresponde a una transformación binaria de la imagen en escala de grises como la mostrada en la Figura 3.4.2-1. Uno de los métodos empleados en Matlab es el de Otsu, este maximiza la varianza entre clases puesto que se basa en la separación de los estadísticos por clases, según los valores de nivel de intensidad que posea cada pixel de la imagen en escala de grises. La idea básica es que un buen umbral de las clases debería ser distinto de dichos valores, mismo que proporciona la mejor separación entre clases, este podría ser el mejor umbral [39][1]. En la Figura 3.4.3-1 puede observarse el efecto de la binarización empleando el método Otsu por medio de la sentencia *imbinarize()* empleada en Matlab, tomando las 25 imágenes en escala de grises de la figura anterior.

Figura 3.4.3-1: Binarización por umbral de las imágenes en escala de grises de la Figura 3.4.2-1



Nombre de la fuente: Elaboración propia

[1] El lector interesado en profundizar sobre dicho método puede consultar este libro y estudiarlo desde la página 764 hasta la 769, ya que aquí no se abordará dado que esta por fuera de los objetivos del proyecto.

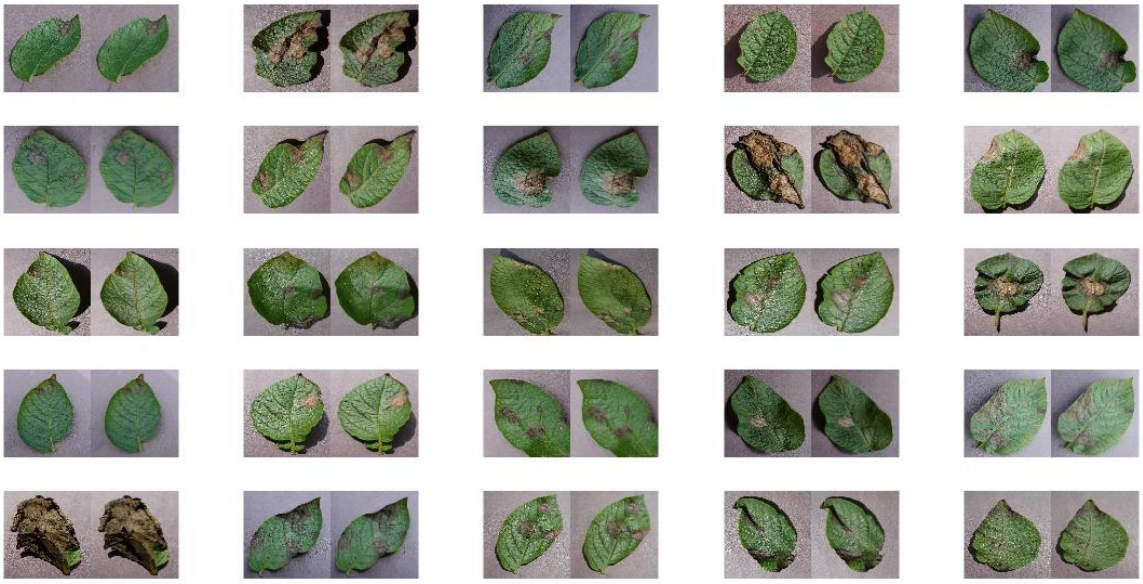
Gracias a esta transformación pueden observarse los artefactos, estos son indeseables y deben ser atenuados cada vez que sea posible. En este trabajo se empleará un filtro espacial como una operación de correlación entre la imagen de entrada y un elemento estructurante en forma de disco de radio unitario, el cual se detalla en la siguiente sesión.

3.4.4 Filtrado espacial de la imagen

El filtro que se ha pensado implementar en la herramienta computacional es descrito por la Ecuación (2.22) y define la sumatoria total de productos en cada pixel de $x(n,m)$ por los de $w(s,t)$. En este caso, $w(s,t)$ representa a un elemento estructurante tipo disco de radio unitario como los hallados en [39, p. 651], comúnmente se conocen como kernels, sus dimensiones suelen ser de números impares y pueden elegirse libremente aunque hay que rellenar de ceros la matriz de entrada. Recuérdese que tanto $x(n,m)$ como $w(s,t)$ son imágenes, para el primer caso $x(n,m) \in \mathbb{R}^{n\times m\times 3}$; $n = m = 256$ es una imagen a color, así $w(s,t) \in \mathbb{R}^{s\times t\times 1}$; $s = t = 3$ es una imagen binaria.

Los kernels que adoptan diversas figuras geométricas hacen parte del procesamiento morfológico de la imagen, pudiendo ser implementado en un filtro espacial por medio de sentencia *fspacial()* de Matlab, el cual permite crear el filtro deseado para el procesamiento de las hojas a color mostradas en la Figura 3.4.2-1, *imfilter()* que es la instrucción encargada de ejecutar el filtro creado. Los resultados de su implementación pueden apreciarse en la Figura 3.4.4-1:

Figura 3.4.4-1: Implementación del filtro espacial para 25 imágenes de entrada



Nombre de la fuente: Elaboración propia

Con el empleo de este filtro se ha logrado obtener una limpieza aceptable de artefactos en cada imagen, el disco de radio unitario se ha usado porque es el que mejor comportamiento tiene sobre la imagen evitando la degradación de la misma, el resultado sobre cada imagen es apreciado en al costado derecho de cada imagen de la Figura 3.4.4-1. Por último, el lector puede replicar este procesamiento empleando el algoritmo mostrado en el Anexo B de este proyecto.

4. Construcción de la red neuronal convolucional

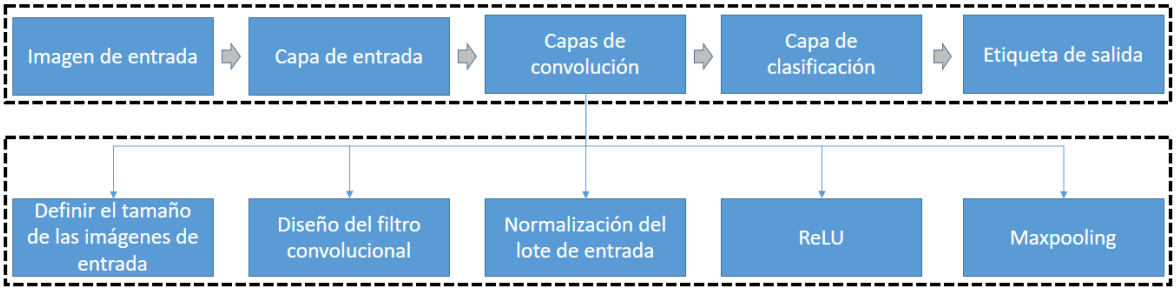
Esta fase del proyecto parte de tres etapas fundamentales para construir la CNN que son: diseño la red, arquitectura de la misma y su fase de entrenamiento. En la primera etapa establece la estructura de las capas de entrada, de convolución, así como de la capa clasificadora. El resultado del diseño ha concebido una arquitectura de red de nueve capas. Por último, se ha escogido el gradiente descendente Adam con el fin de acelerar la convergencia al valor mínimo local buscado durante su entrenamiento, reduciendo el tiempo que ella gasta en ese proceso.

4.1 Diseño de la red

Antes de construir la red neuronal que aborde el problema de clasificar las hojas de papa con tizón tardío es importante estimar algunos parámetros de la misma, lo cual involucra una tarea de diseño. Con el fin de diseñar la ConvNet de este proyecto, fue necesario tomar las recomendaciones de [33], [28], [30] y [32], para implementar la capa de entrada, las capas de convolución y la capa clasificadora. El objetivo de esta fase es el de diseñar una red neuronal convolucional con al menos 100.000 parametros para entrenamiento, que no supere diez capas en su arquitectura, usando el algoritmo Adam para su entrenamiento y que sea fácil de construir en el entorno de Matlab.

Por lo anterior, se ha propuesto la metodología representada mediante el diagrama de bloques de la Figura 4.1-1:

Figura 4.1-1: Diagrama de bloques que conforma la metodología del diseño



Nombre de la fuente: Elaboración propia

A continuación, serán descritas cada uno de los bloques mencionados en el diagrama ilustrado en la Figura 4.1-1:

4.1.1 Imagen de entrada

Cada imagen está definida por dos clases que son: hojas de papa sana (C_1) y enfermas de tizón tardío (C_2), sus dimensiones de entrada son de $256 \times 256 \times 3$, ellas no han sido pre – procesadas para el entrenamiento de la CNN.

4.1.2 Capa de entrada

Está determinada por la cantidad de entradas que van a ser ingresadas a la red. Para este caso se dispone de dos entradas, ello sugiere la conformación de dos subconjuntos ligados a dos clases C_1 y C_2 (véase la sesión 2.2), donde la primera corresponde a la etiqueta que identifica las hojas sanas y la segunda a la de hojas enfermas con tizón tardío. Recuérdese que $C_1 \subseteq H_1 \wedge C_2 \subseteq H_2 \rightarrow H_1 \cup H_2 = H$, los subconjuntos H_1 y H_2 están coformados por las hojas de las hortalizas de papa sana y con Tizón Tardío respectivamente, ambos poseen la misma cantidad de elementos. La clase C_1 corresponde a la etiqueta hojas sanas y C_2 a enfermas con Tizón Tardío.

4.1.3 Capas de convolución

Estas capas son quizás las más importantes de toda la red porque cada una va extrayendo la información más relevante de la imagen obteniendo características de bajo y alto nivel, conforme la imagen circula a través de cada capa convolucional. Su construcción involucra las dimensiones de entrada de la imagen, los filtros convolucionales, la normalización de lote o Batch Normalization, la función de activación, así como la de agrupamiento.

4.1.4 Dimensiones de entrada

Las dimensiones de la imagen de entrada se han fijado de $256 \times 256 \times 3$ a través de la instrucción *imageInputLayer()* la cual restringe a dichas dimensiones los patrones de entrada $\boldsymbol{X}_p$, que para este caso son las imágenes de entrada para el entrenamiento y prueba de la red.

4.1.5 Diseño del filtro convolucional

Este filtro ha sido diseñado teniendo en cuenta la estimación de los pasos (stride) en la operación de la correlación espacial de las imágenes de entrada a la red, el relleno de ceros en la imagen de entrada y el tamaño de cada kernel el cual conforma el filtro convolucional de cada una de estas capas. Es importante resaltar que las dimensiones de la matriz de salida deben conservar el ancho y alto de la imagen de entrada. Para satisfacer lo anterior debe recurrirse a la Ecuación (4.1) [33, p. 169]:

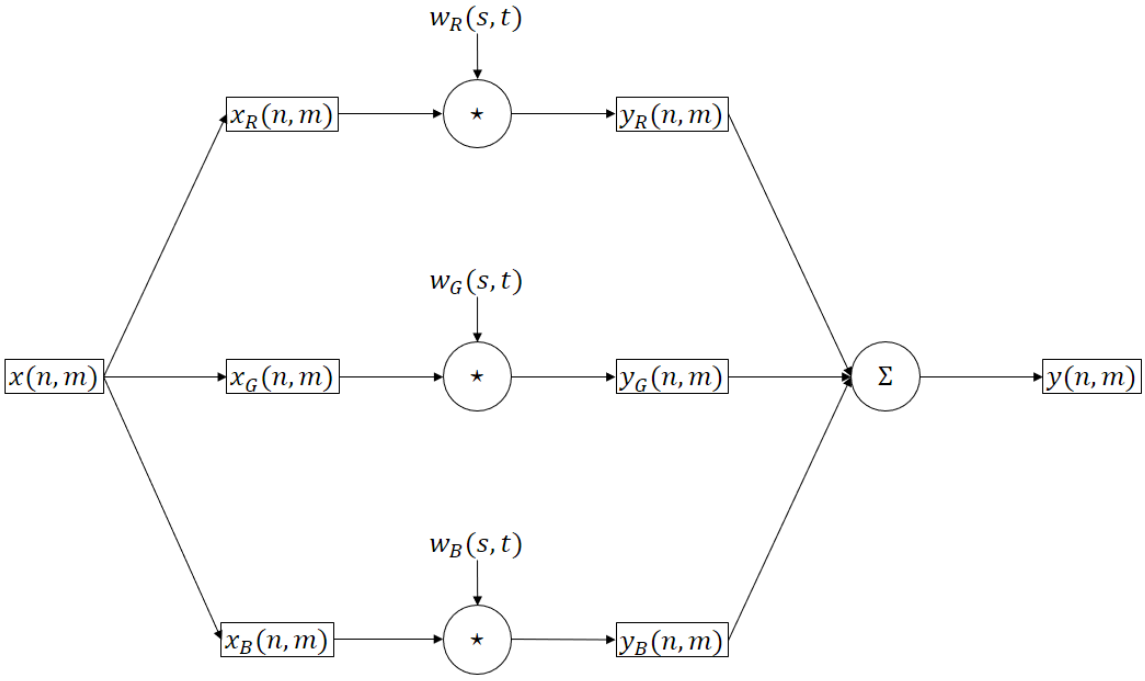
$$D_o^{im} = \frac{D_i^{im} - F + 2P}{S} + 1 \tag{4.1}$$

Donde D_o^{im} es el tamaño de la matriz de salida, D_i^{im} es el tamaño de la imagen de entrada, F es el tamaño del kernel, comúnmente, lo conforma una matriz de 3×3, 5×5 o 9×9, la cantidad de relleno de ceros o zero pading es designada por P, siendo S el número de pasos o strides que toma el desplazamiento horizontal del kernel en la convolución espacial. Una buena estimación del relleno de ceros está dada por la Ecuación (4.2):

$$P = \frac{F-1}{2} \tag{4.2}$$

Este proyecto ha contemplado operar filtros convolucionales con kernels cuadrados de 5×5 como criterio de diseño, puesto que aumenta la cantidad de parámetros entrenables en la red, evita un excesivo rellenado de ceros en la imagen de entrada $x(n,m)$ lo cual conlleva a un moderado consumo computacional en la fase de entrenamiento. Claramente se observa que $P = 2$, considerando que cada imagen de la base de datos PlantVillage posee un tamaño de 256×256, D_o^{im} también debe conservarlo. Tomando un paso deslizante $S = 1$ para el desplazamiento del kernel en la matriz de entrada y usando la ecuación (4.1) la matriz de salida satisface la condición anterior, puesto que $D_o^{im} = D_i^{im} = 256$. La Figura 4.1.5-1 muestra el esquema que describe el diseño del filtro convolucional usando el cual cumple con la condición anteriormente mencionada.

Figura 4.1.5-1: Esquema del filtro convolucional



Nombre de la fuente: Elaboración propia

Este nodo de convolución atribuye 76 parámetros que deben ser entrenados que son: 25 conexiones sinápticas otorgadas por cada kernel $w_R(s,t)$, $w_G(s,t)$ y $w_B(s,t)$, dado que son tres (uno por cada componente de color) entonces hay 75 conexiones que deben ser

entrenadas, adicionalmente, debe considerarse que el bias también es actualizado en la fase de entrenamiento razón por lo cual es un parámetro de este tipo, mismo que hace parte de la neurona de convolución. Por último, mientras $x(n,m) \in \mathbb{R}^{256\times256\times3}$, $x_R(n,m), x_B(n,m) \wedge x_G(n,m) \in \mathbb{R}^{256\times256\times1}$ al igual que $y(n,m), w_R(s,t), w_B(s,t) \wedge w_G(s,t) \in \mathbb{R}^{5\times5\times1}$ representan los kernels de la Figura 4.1.5-1 los cuales van a ser correlacionados con las matrices de entrada, dicha correlación es simbolizada con el operador ($\star$).

4.1.6 Normalización de lote – Batch Normalization

La normalización del lote permite evitar la distorsión de la información cuando esta viaja a través de todas las capas de la red siendo una medida estadística, la media está dada como sigue:

$$\mu_B = \frac{1}{N}\sum_{i=0}^{N} y_i(n,m) \tag{4.3}$$

Donde μ_B es valor promedio del lote, N es el tamaño del minilote y $y_i(n,m)$ activacion del filtro convolucional ante el i – ésimo patrón del minilote. La varianza del lote σ_B^2 la describe la Ecuación (4.4):

$$\sigma_B^2 = \frac{1}{N}\sum_{i=0}^{N}(y_i(n.m) - \mu_B)^2 \tag{4.4}$$

La normalización del lote normaliza cada elemento de $y_i(n.m)$ calculando su media y varianza, para ello suele usarse las ecuaciones (4.3) y (4.4). Dicha normalización es escrita como sigue:

$$y_i^{norm}(n,m) = {(y_i(n,m) - \mu_B)}\Big/{\sqrt{\sigma_B^2 + \varepsilon}} \tag{4.5}$$

Donde ε es una constante usada para mejorar la estabilidad numérica cuando σ_B^2 es muy pequeña. Así la salida de la capa normalizada está dada por la Ecuación (4.6):

$$y_i^{Bnorm} = \gamma \cdot y_i^{norm}(n,m) + \eta \tag{4.6}$$

Donde η es un valor de offset o ajuste y γ es un parámetro de aprendizaje que es actualizado por el entrenamiento de la red.

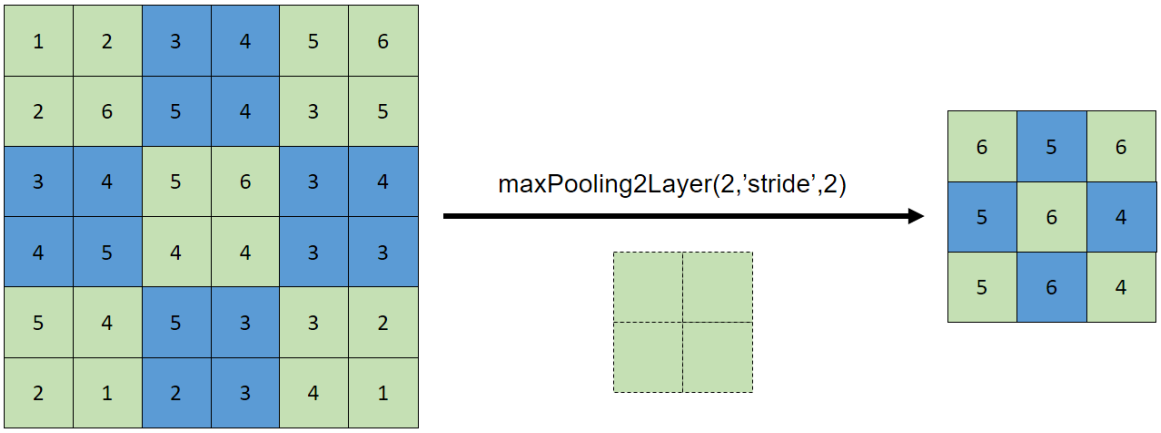
4.1.7 Función de activación ReLU

La activación es la función ReLU estudiada en el segundo capítulo de este trabajo y consiste en llevar a cero cada elemento cuya magnitud sea negativa en la matriz de salida.

4.1.8 Función de agrupamiento - Maxpooling

La operación de agrupamiento reduce las dimensiones de los mapas de características usando maxpooling la cual extrae los parches de los mapas de características. Su funcionamiento puede verse en la Figura 4.1.8-1:

Figura 4.1.8-1: Operación maxpooling



Nombre de la fuente: Elaboración propia

Para el diseño de la ConvNet se ha considerado un agrupamiento cuyo tamaño es una selección de 2×2 con paso de 2. La función de agrupamiento selecciona un área de 2×2 pixeles en el mapa de características formando una matriz cuadrada de este tamaño, luego se extrae el pixel de mayor valor, tal como se aprecia en la Figura 4.1.8-1. Después ocurre un desplazamiento de dos pixeles en sentido horizontal (izquierda a derecha) formando una nueva matriz donde nuevamente se escoge el pixel que contiene el máximo

valor. El proceso se repite reiteradamente hasta cubrir el área total del mapa de características conllevando una reducción de su tamaño a la mitad, por lo cual, esta operación es vital para arquitectura de la red.

4.1.9 Capa clasificadora

Esta capa está configurada por tres operaciones que son: la Full Connect Layer (FCL), la función de activación llamada *Softmax* y la capa de clasificación. FCL es la capa encargada de la combinación lineal definida por la Ecuación (2.6), entre las neuronas de la última capa oculta y las neuronas de la capa de salida. La activación *Softmax* es la neurona que normaliza la combinación lineal de la FCL produciendo una sumatoria de datos produciendo un rango de $[0,1]$. Dicha función está dada por la Ecuación (4.7):

$$f_{softmax}(\boldsymbol{W}^T\boldsymbol{X}) = e^{\boldsymbol{W}^T\boldsymbol{X}+b} \Big/ \sum_{i=1}^{N} e^{\boldsymbol{W}_i^T\boldsymbol{X}_i+b_i} \tag{4.7}$$

La capa de clasificación evalúa la función de costo representada por la Ecuación (2.39), la clasificación infiere el número de clases del tamaño de salidas de la capa previa. Para el diseño y construcción de la CNN se implementan estas capas para conectar las neuronas convolucionales a la capa de salida. El diseño de la red ha originado una ConvNet con 7 capas de convolución, una capa de entrada y otra de salida, con un total de 26.631 parámetros para entrenar. En la siguiente sesión de este capítulo se muestra la arquitectura de la red, la cantidad de filtros por capa, el tamaño de los *kernels*, el relleno de ceros, entre otros.

4.1.10 Etiquetas de salida

Se atribuye al resultado de clasificación perteneciente a la primera o segunda clase según la imagen de entrada hacia la red neuronal.

4.2 Arquitectura

El diseño de la red fue implementado en Matlab R2018 con el fin de crear la CNN la cual permite el diagnóstico de hojas sanas y con Tizón Tardío. La imágenes RGB de la base de datos PlantVillage son de dimensiones $256 \times 256 \times 3$ y constituyen las entrada de la CNN. Hay dos etiquetas creadas en la base de datos; Papa_sana y Papa_con_Tizón_Tardío, por lo cual, ellas configuran solo dos entradas en dicha red. La primera capa convolucional contiene 32 filtros espaciales, una capa de normalización de lote, otra de activación ReLU, así como una para el agrupamiento. Por consiguiente, las capas L_1 a L_7 poseen filtros de 32, 64, 64, 128, 128, 256, 256, niveles respectivamente.

El cambio de dimensión de la imagen en cada salida está dado gracias a la función maxpooling generando mapas de características de $256 \times 256 \times 32$, $128 \times 128 \times 64$, $64 \times 64 \times 64$, $32 \times 32 \times 128$, $16 \times 16 \times 128$, $8 \times 8 \times 256$, y de $4 \times 4 \times 256$, respectivamente. La capa de conexión completa L_8 posee dos neuronas de salida e incluye las operaciones descritas en la sesión 4.1.3. La arquitectura de la ConvNet implementada en Matlab se muestra en la Figura 4.2-2:

Figura 4.2-2: Implementación de la ConvNet en Matlab

```
>> Red.Layers

ans =

  32x1 Layer array with layers:

     1   'imageinput'    Image Input             256x256x3 images with 'zerocenter' normalization
     2   'conv_1'        Convolution             32 5x5x3 convolutions with stride [1  1] and padding [2  2  2  2]
     3   'batchnorm_1'   Batch Normalization     Batch normalization with 32 channels
     4   'relu_1'        ReLU                    ReLU
     5   'maxpool_1'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
     6   'conv_2'        Convolution             64 5x5x32 convolutions with stride [1  1] and padding [2  2  2  2]
     7   'batchnorm_2'   Batch Normalization     Batch normalization with 64 channels
     8   'relu_2'        ReLU                    ReLU
     9   'maxpool_2'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
    10   'conv_3'        Convolution             64 5x5x64 convolutions with stride [1  1] and padding [2  2  2  2]
    11   'batchnorm_3'   Batch Normalization     Batch normalization with 64 channels
    12   'relu_3'        ReLU                    ReLU
    13   'maxpool_3'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
    14   'conv_4'        Convolution             128 5x5x64 convolutions with stride [1  1] and padding [2  2  2  2]
    15   'batchnorm_4'   Batch Normalization     Batch normalization with 128 channels
    16   'relu_4'        ReLU                    ReLU
    17   'maxpool_4'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
    18   'conv_5'        Convolution             128 5x5x128 convolutions with stride [1  1] and padding [2  2  2  2]
    19   'batchnorm_5'   Batch Normalization     Batch normalization with 128 channels
    20   'relu_5'        ReLU                    ReLU
    21   'maxpool_5'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
    22   'conv_6'        Convolution             256 5x5x128 convolutions with stride [1  1] and padding [2  2  2  2]
    23   'batchnorm_6'   Batch Normalization     Batch normalization with 256 channels
    24   'relu_6'        ReLU                    ReLU
    25   'maxpool_6'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
    26   'conv_7'        Convolution             256 5x5x256 convolutions with stride [1  1] and padding [2  2  2  2]
    27   'batchnorm_7'   Batch Normalization     Batch normalization with 256 channels
    28   'relu_7'        ReLU                    ReLU
    29   'maxpool_7'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
    30   'fc'            Fully Connected         2 fully connected layer
    31   'softmax'       Softmax                 softmax
    32   'Salida'        Classification Output   crossentropyex with classes 'Papa_Sana' and 'Papa_con_Tizon_Tardio'
>>
```

Nombre de la fuente: Elaboración propia

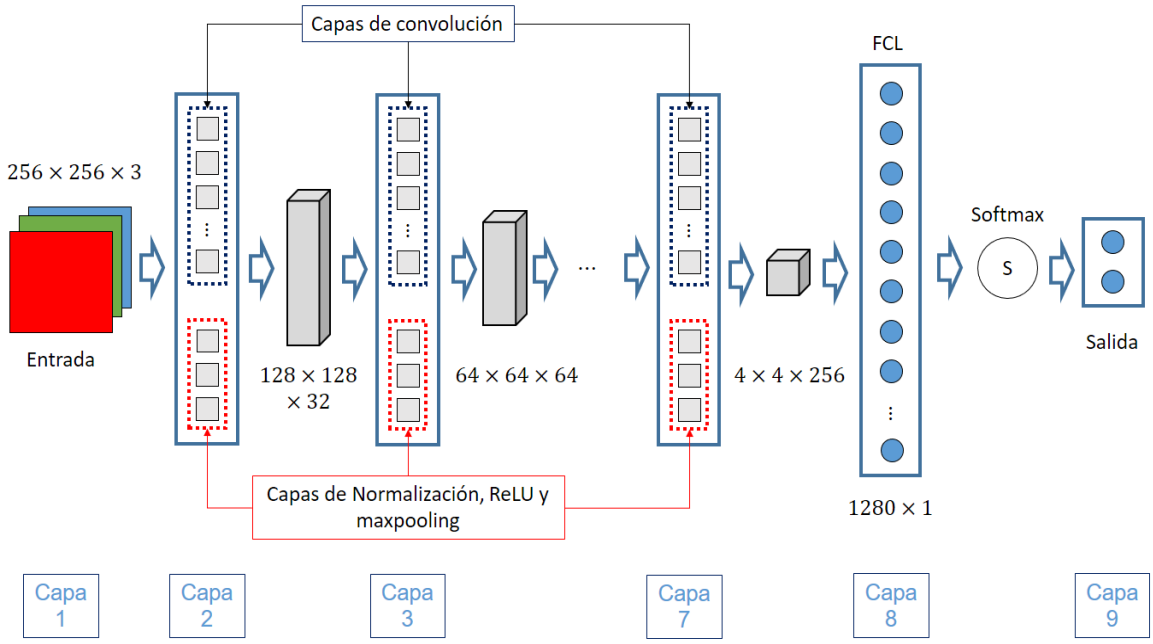
Ella contiene 26.631 parámetros de entrenamiento, siendo suficientes para redes de este tipo. La Tabla 4-1 muestra detalladamente las cualidades para obtener dichos parámetros.

Tabla 4-1: Cualidades de la CNN diseñada.

No. de capas	D_i^{im}	F	Cantidad	S	P	D_0^{im}	Parámetros por capa	Total de parámetros
1	256×256	5×5	32	1	2	256×256	2.432	1'598.466
2	128×128	5×5	64	1	2	128×128	51.264	
3	64×64	5×5	64	1	2	64×64	102.464	
4	32×32	5×5	128	1	2	32×32	204.928	
5	16×16	5×5	128	1	2	16×16	409.728	
6	8×8	5×5	256	1	2	8×8	819.456	
7	4×4	5×5	256	1	2	4×4	8.194	

Se ha notado que el aumento o decremento en la totalidad de parámetros a entrenar está en función del tamaño del kernel F así como de la cantidad de filtros convolucionales, causando variabilidad del costo computacional. Una forma de reducirlo es aumentando el número de capas y reduciendo el tamaño de los filtros, como es el caso de las arquitecturas de Resnet e Inception [33]. La arquitectura de la esta red fue basada en la red VGG-16 pero con el empleo de filtros 5×5, misma que se muestra en la Figura 4.2-3:

Figura 4.2-3: Arquitectura de la CNN



Nombre de la fuente: Elaboración propia

4.3 Fase de entrenamiento

En esta fase se ha escogido el gradiente descendente Adam, con el fin de aprovechar la bondad de acelerar la convergencia al valor mínimo local buscado durante el entrenamiento de la red, optimizando el tiempo que ella gasta en ese proceso. La velocidad de la dinámica en la actualización de los pesos sinápticos y los bias es aumentada debido al efecto que causan los momentos $M_1(t) \wedge M_2(t)$ aportando a lo anteriormente mencionado, las propiedades del entrenamiento de la red que se ha diseñado son mostradas en la Tabla 4-2:

Tabla 4-2: Especificaciones del entrenamiento.

Propiedad	Valor
ρ_1	0.9000
ρ_2	0.9990
ε	1×10^{-8}
α_0	1×10^{-3}
Regularización $\mathbb{L}_2$	1×10^{-4}
Método del umbral del gradiente	Norma $\mathbb{L}_2$
Valor máximo de épocas (*epoch*)	100
Tamaño del minilote	38

Las propiedades ρ_1, ρ_2, ε y α_0 fueron estudiados en la sesión 2.4.6 de este trabajo, la regularización y el método de ajuste de umbral se basan en la norma $\mathbb{L}_2$, su estudio está fuera del alcance de este proyecto, para mayor información refiérase a [47]. Una época es el número de veces que el lote pasa por la red, se ha concebido 12 iteraciones por época, esto implica que cada lote pasa por la ConvNet 1200 veces. El tamaño del minilote sirve como muestra para que el algoritmo ajuste los pesos y los bias en la fase de entrenamiento, la cual tuvo una durabilidad de 3 horas aproximadamente. La red ha sido entrenada con 1304 imágenes de la base de datos PlantVillage, 304 de ellas corresponden a hojas sanas, siendo el restante atribuidas al tizón tardío.

5. Elaboración de la herramienta computacional

La herramienta computacional involucra una red neuronal convolucional la cual se entrenó para obtener la clasificación de hojas de papa de dos clases; sanas u con tizón tardío. Las hojas con esta enfermedad indican que el herbáceo padece una morbilidad crónica que puede ser catastrófica en los cultivos de papa de gran cantidad de hectáreas, por ello, surge la motivación de hacer una herramienta computacional o software (SW) para fijar un avance tecnológico en el sector agrícola, principalmente para los agricultores. En este capítulo se aborda su elaboración a partir de tres enfoques que son; el diseño de la interfaz de usuario, su arquitectura computacional, y su construcción basada en los enfoques anteriores.

5.1 Diseño de la interfaz grafica

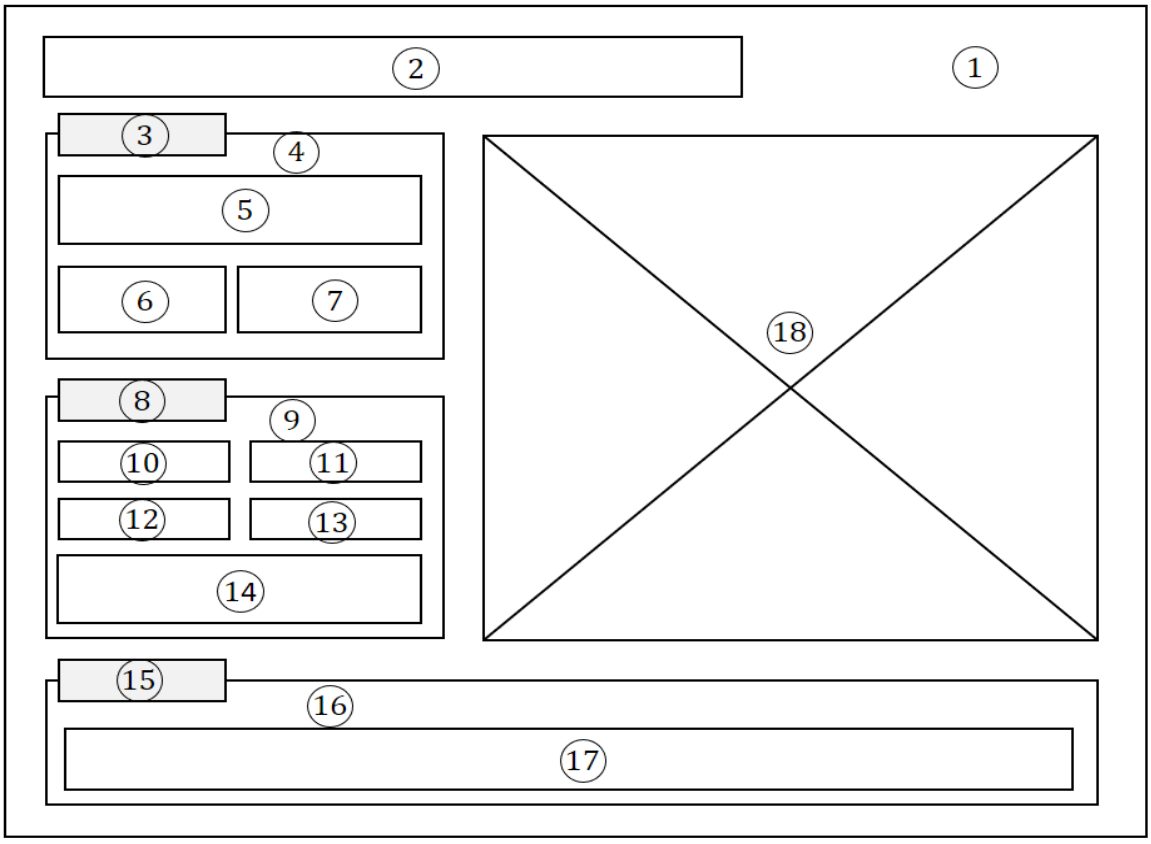
Para el diseño de la interfaz de usuario es necesario contemplar los materiales y métodos a utilizar para tal fin. En primer lugar, se ha elegido la plataforma de desarrollo llamada GUIDE de Matlab, dada su factibilidad para integrar la CNN, los filtros espaciales, la identificación de los artefactos, y demás aspectos mencionados en el capítulo anterior. De este modo, la lista de materiales y métodos contemplados para el diseño y construcción de la herramienta está dada en la Tabla:

Tabla 5-1: Materiales y métodos para diseñar y construir la herramienta.

Tipo		Recurso	Utilidad en la herramienta
Material	**Método**		
X		Computadora portátil marca HP	Usada para ejecutar el entorno de desarrollo, crear la red neuronal, elaborar la herramienta computacional, verificar su funcionamiento a partir de las bases de datos, entre otros.
X		Entorno Matlab R2018	Es la plataforma de desarrollo para crear el SW así como la ConvNet, permite organizar y administrar las bases de datos, ofrece potentes algoritmos para implementar filtros espaciales.
X		Microsoft Power Point	SW usado para elaborar los bocetos de la presentación de la interfaz gráfica de usuario.
X		Red neuronal convolucional	Creada para dotar a la herramienta de un mecanismo de clasificación de las hojas.
X		Base de datos	Es la base del proyecto puesto que permite entrenar la CNN y evaluar el desempeño de la herramienta computacional.
X		Documentación técnica	De gran utilidad al momento de resolver problemas técnicos, dicha documentación no hace parte de las referencias bibliográficas.
	X	Structural Similarity Index (SSIM)	Permite valorar las imágenes de entrada con respecto a una de referencia.
	X	Filtro espacial estudiado en la sesión 3.4.3	Permite separar los artefactos de una imagen contaminada.
	X	Pruebas de ensayo / error	Permite identificar los problemas de la herramienta ya construida.

Aunque es esta sesión no adquiere tanta relevancia el listado de la tabla anterior, dichos recursos si serán de vital importancia en la arquitectura de la herramienta asi como en su construcción. El diseño de la interfaz gráfica de usuario configura el aspecto visual que tendrá el SW, mismo que es mostrado en la Figura 5.1-1:

Figura 5.1-1: Diseño de la interfaz gráfica de usuario de la herramienta computacional



Nombre de la fuente: Elaboración propia

Cada elemento de la ilustración anterior es enumerado por la Tabla 5-2.

Tabla 5-2: Elementos que componen la interfaz gráfica de usuario.

Número	Nombre	Número	Nombre	Número	Nombre
1	Fondo	7	Texto dinámico “Resultado numérico en %”	13	Botón “Filtro”
2	Título principal	8	Subtitulo “Menú de usuario”	14	Botón “Clasificar”
3	Subtitulo “Diagnostico”	9	Panel de botones	15	Subtitulo “Sobre el autor”
4	Panel de texto	10	Botón “Cargar”	16	Panel de texto
5	Texto dinámico de etiquetas	11	Botón “Aceptar”	17	Texto estático, ofrece información del autor
6	Texto estático ”Exactitud”	12	Botón “Artefactos”	18	Pantalla o panel de visualización

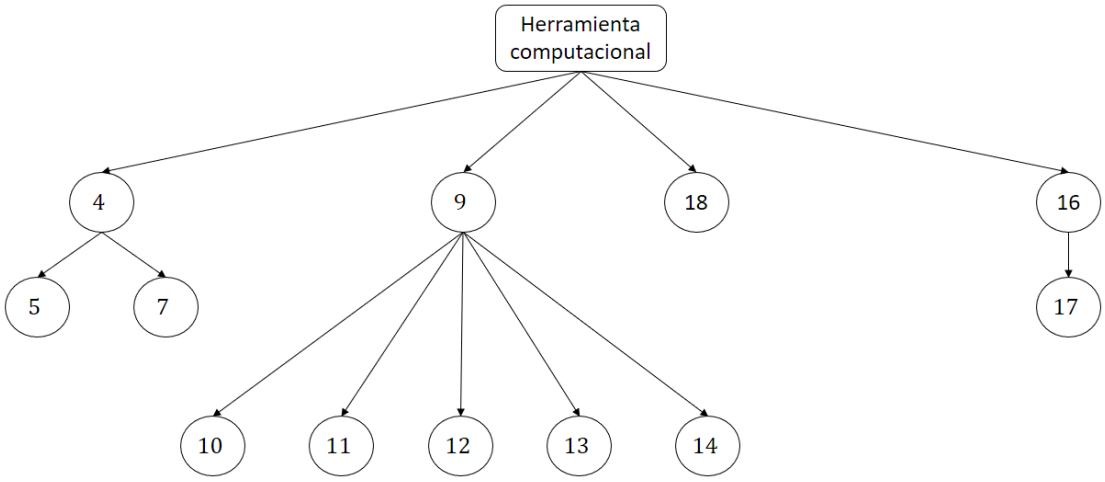
Su descripción puede hallarse en la Tabla 5-3:

Tabla 5-3: Especificaciones de los elementos usados en la interfaz.

Elemento	Tipo	Descripción
1	Fondo	Establece el aspecto de fondo del SW
2, 3, 6, 8, 15, 17	Texto estático	Configura las zonas que contienen texto fijo como es el caso de: el título de la herramienta, los subtítulos de cada sub área y descripciones como información del autor y el nivel de exactitud.
4, 9, 16	Panel de texto	Agrupa todos los textos estáticos y dinámicos que posee la herramienta computacional.
5, 7	Texto dinámico	Muestra la información numérica y de texto como el porcentaje de exactitud en la clasificación de las hojas y su etiqueta. Además “5” muestra indicaciones de ayuda para el usuario.
10, 11, 12, 13, 14	Botón	Aplican las funciones computacionales; en su orden, carga de imagen, aceptar la misma como entrada, ilustración de artefactos, filtro, y clasificación de la imagen aceptada.
18	Pantalla (Axes)	Es el área designada para visualizar la imagen de entrada, sus artefactos y la salida del filtro.

Los elementos de la Tabla 5-3 son de vital importancia para el funcionamiento de la herramienta computacional, en la Figura 5.1-2 puede apreciarse el mapa de navegación del software.

Figura 5.1-2: Mapa de navegación



Nombre de la fuente: Elaboración propia

El mapa anterior es bastante simple de interpretar, por ejemplo: al costado derecho de la figura, los números que están encerrados en círculos son 4, 5 y 7. Observando la Tabla 5-2 se aprecia que el primero identifica el panel de texto que contiene el resultado de la hoja de entrada producto de la clasificación. El número 5 asocia el texto dinámico que

muestra la etiqueta de clasificación, es decir, “Hoja sana” u “Hoja con Tizón Tardío”. Finalmente, el número 7 informa el nivel de exactitud alcanzado en dicha clasificación en un margen que va desde 0 hasta 100%. Análogamente, puede interpretarse el menú de usuario, la visualización en la pantalla, así como la información del autor.

5.2 Arquitectura del software

En la arquitectura se describen cada uno de los elementos que componen la herramienta computacional, los cuales pueden ser divididos en tres modalidades que son: visual o de apariencia, operativa o de usuario y lógica – computacional.

5.2.1 Modalidad visual o de apariencia

Modalidad visual o de apariencia: se estructura del fondo, los textos dinámicos y estáticos, los paneles de texto y de botones, así como la pantalla de visualización donde el usuario puede observar la imagen cargada como entrada para luego ser clasificada.

5.2.2 Modalidad operativa o de usuario

Modalidad operativa o de usuario: esta descrito por el mapa de navegación de la Figura 5.1-2, la interfaz está elaborada para que sea fácil de usar, siendo intuitiva y amigable para el usuario.

5.2.3 Modalidad lógica – computacional

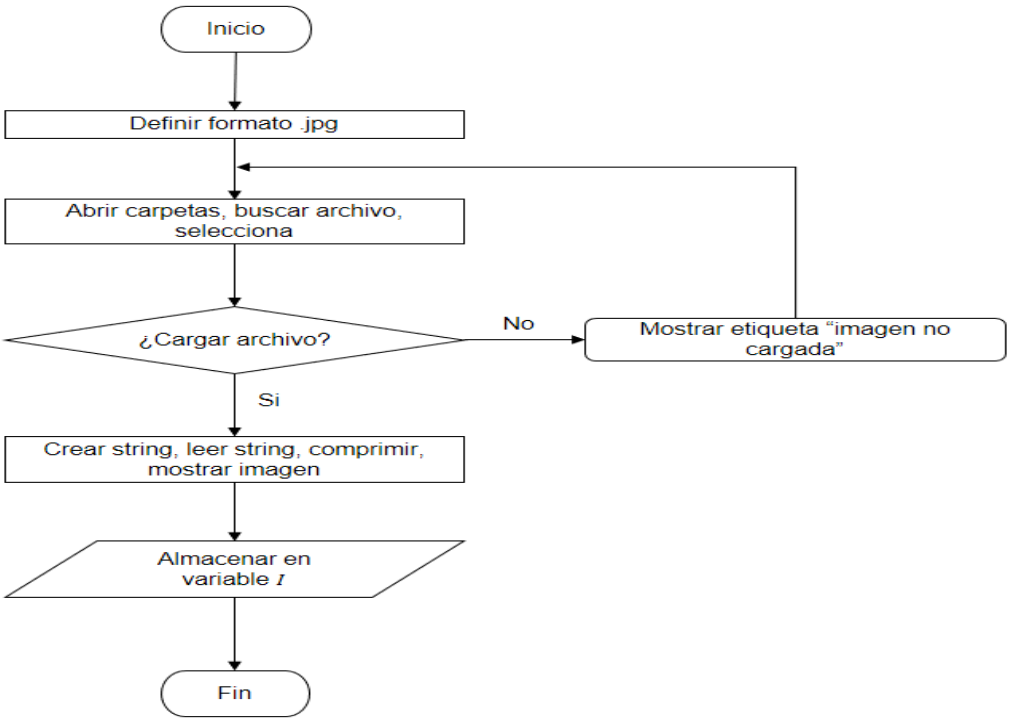
Es el corazón de esta herramienta, se compone de múltiples funciones que permiten efectuar operaciones matemáticas, establecer reglas o secuencias, y mostrar resultados frente a las imágenes de entrada.

Inicio o arranque del software: fija todas las variables globales anteriormente programadas, configura la presentación de la pantalla, el texto, el fondo, así como de los

botones. Allí también es cargada la red neuronal convolucional, como parte integral de la herramienta para el diagnóstico de las hojas de plantas de papa.

Carga de una imagen: esta función está a cargo del botón “Cargar”, mismo que permite efectuar varias tareas como buscar, aceptar o cancelar la imagen de entrada. Desde el punto de vista computacional la imagen es vista como una matriz de elementos de dimensiones $n \times m \times 3$, su orden es similar al de la Ecuación (2.23), misma que se interpola a una resolución de $256 \times 256 \times 3$ mediante la función *imresize()* de Matlab, ella aplica el método Nearest Neighbor Interpolation (NNI) para reasignar cada pixel de la imagen. Adicionalmente, los datos de la matriz son almacenados en una variable la cual será contenida en el banco de datos del objeto configurado por la función del botón cargar. El diagrama de flujo mostrado en la Figura 5.2.3-1 ilustra la funcionalidad del botón cargar.

Figura 5.2.3-1: Diagrama de flujo del botón “Cargar”



Nombre de la fuente: Elaboración propia

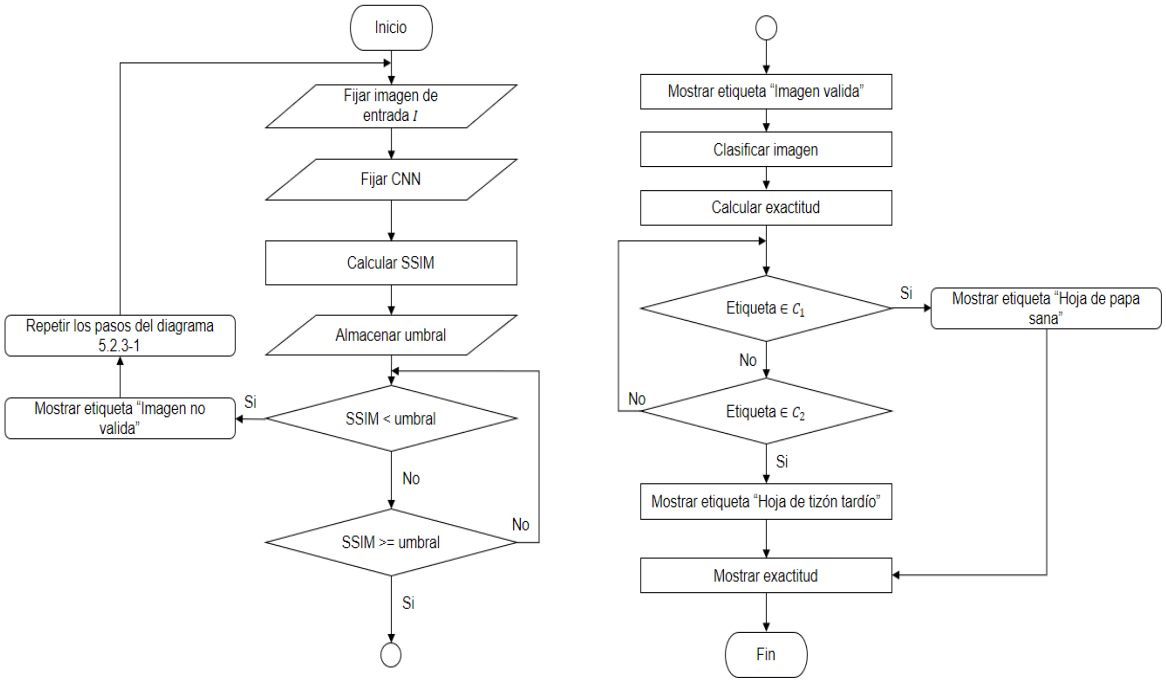
Establecer imagen como entrada: esta función es responsabilidad del botón “Aceptar”, su secuencia es simple pues llama a la variable I y la muestra como imagen en la pantalla, a su vez la guarda en la variable I_1, la cual será archivada como entrada para la CNN.

Visualización de los artefactos de la imagen: está a cargo del botón “Artefactos” e involucra cálculos computacionales concernientes a la convolución espacial y resta de matrices, su resultado es acumulado como una variable local y mostrado en la pantalla para visualización de imágenes.

Filtrado de la imagen: el botón “Filtro” aplica la separación de artificios no deseados en la imagen mediante el filtrado espacial, asimismo, fija la variable I_1 como imagen filtrada de entrada hacia la red convolucional.

Clasificación de la imagen: el botón “Clasificar” identifica la similitud de la imagen de entrada dada una imagen de referencia. Al pasar por este algoritmo, ella es ingresada como entrada a la red, la cual se encarga de evaluar la pertenencia de la imagen hacia cualquiera de las clases C_1 o C_2 generando la etiqueta de salida según su función de decisión. Dicha etiqueta pasa por un algoritmo de asignación, el cual agrega los contenidos de salida para los textos dinámicos 5 y 7, que se traduce en el diagnóstico de la hoja. El diagrama de flujo asociado a este botón puede apreciarse en la Figura 5.2.3-2:

Figura 5.2.3-2: Diagrama de flujo del botón “Clasificar”

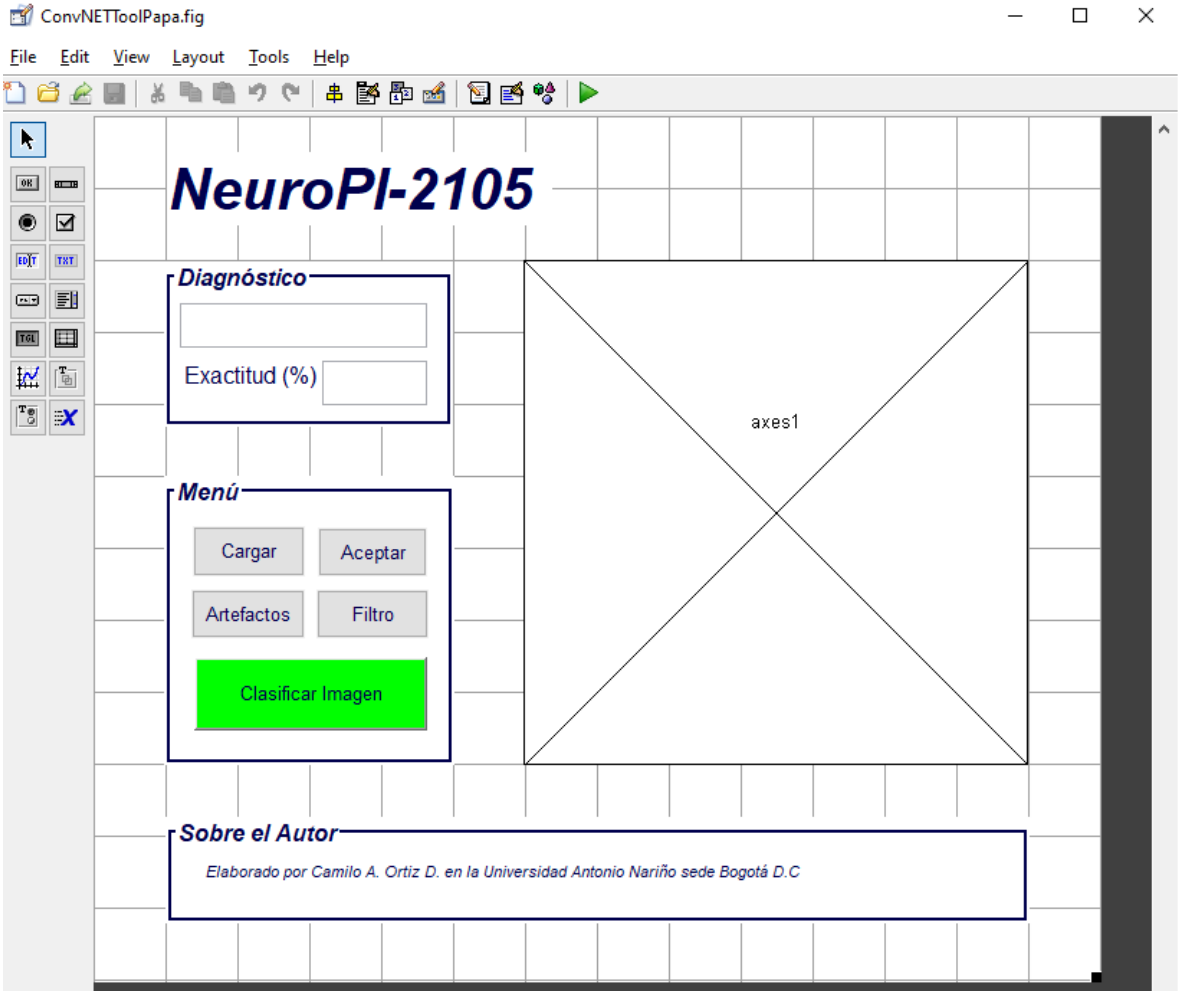


Nombre de la fuente: Elaboración propia

5.3 Construcción de la herramienta

La construcción de la herramienta computacional es elaborada mediante la plataforma GUIDE de Matlab siguiendo el bosquejo mostrado en la Figura 5.1-1. Al implementar los títulos estáticos y dinámicos, los botones, la pantalla de visualización, entre otros, la apariencia de la herramienta computacional deriva en la imagen de la Figura 5.3-1:

Figura 5.3-1: Apariencia de la herramienta computacional

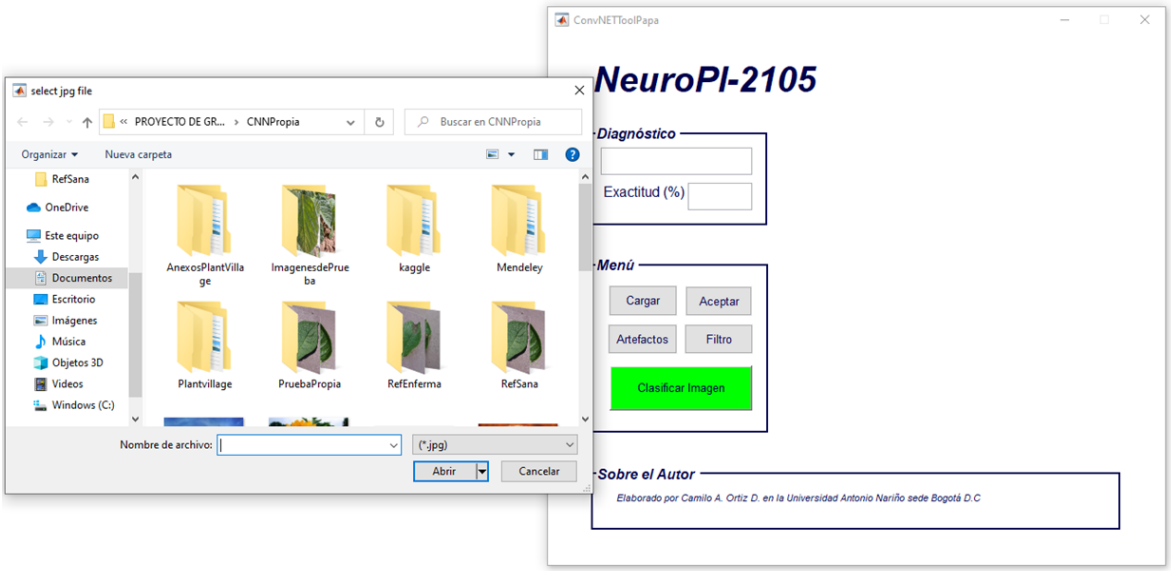


Nombre de la fuente: Elaboración propia

Nótese que la herramienta es llamada "*NeuroPI-2105*" su nombre representa la unión de tres aspectos: la inteligencia artificial, la enfermedad a diagnosticar en las hojas de papa y la fecha de creación. La palabra "*Neuro*" yace de la integración de neuronas artificiales en

el SW, en este caso, dichas neuronas se atribuyen a las capas convolucionales de la red incorporada. La abreviatura "*PI*" designa el nombre científico y epidemiológico que tiene la enfermedad, es decir, Phytophthora Infestans. Por último, su fecha de creación está establecida para el mes de mayo del 2021. Con el fin de ilustrar la operatividad de *NeuroPI-2105*, se hace alusión a las siguientes imágenes, ellas muestran el funcionamiento de la herramienta, así, por ejemplo, la Figura 5.3-2 refleja la acción de pulsar el botón "Cargar":

Figura 5.3-2: Acción de pulsar el botón Cargar

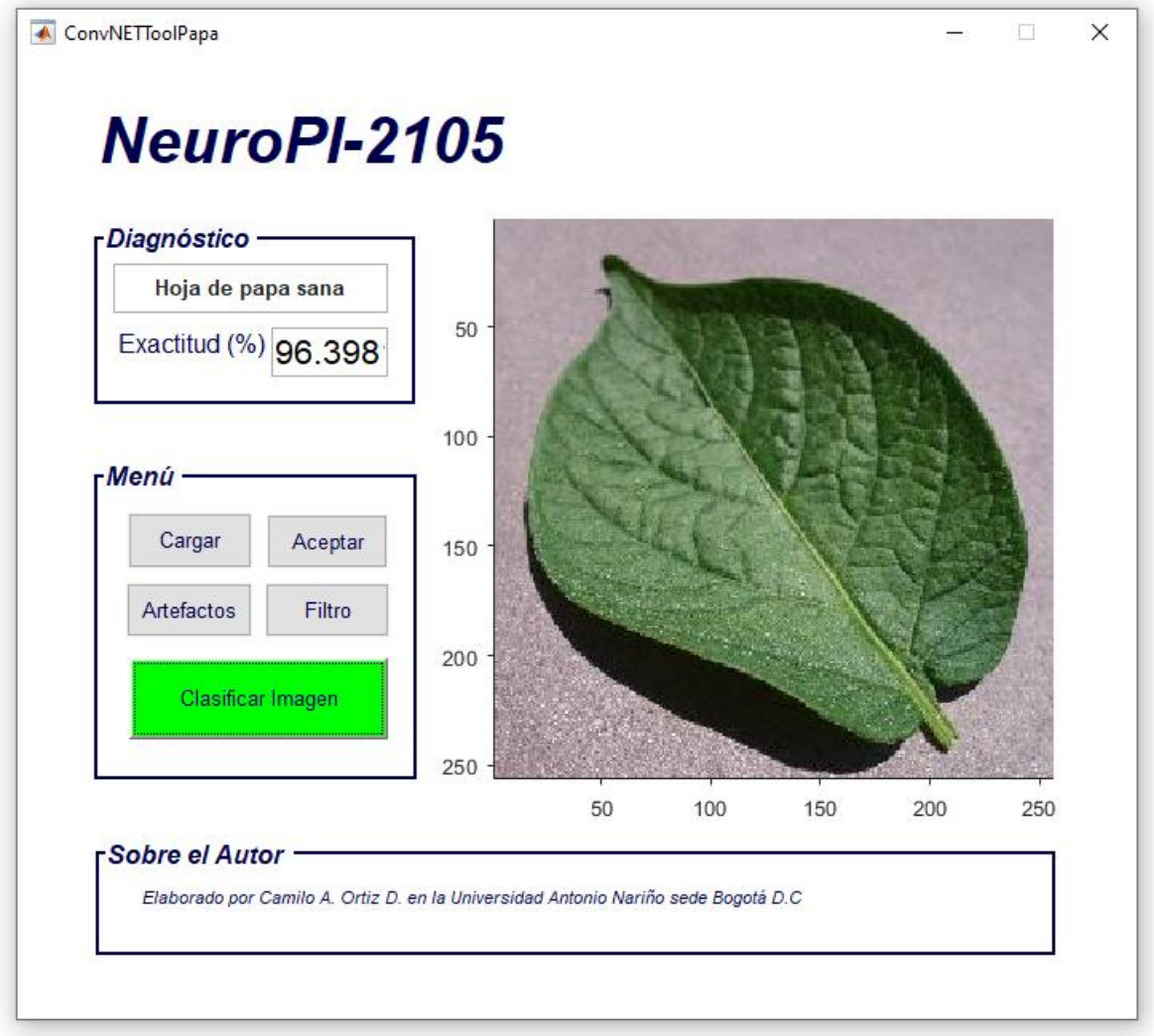


Nombre de la fuente: Elaboración propia

El texto dinámico ubicado al costado superior del panel de diagnóstico indica los mensajes visuales que el agricultor tiene como ayuda para usar la herramienta. Al presionar "Aceptar" se configura una imagen en la *NeuroPI-2105* como una señal de entrada que va hacia la CNN, asimismo, permanecen las indicaciones el panel de diagnóstico con el fin de orientar al usuario. El panel de diagnóstico muestra dos tipos de datos: cualitativo y cuantitativo. El primero corresponde a la etiqueta mostrada en la caja de texto dinámico ubicado por encima del cuadro más pequeño, define la cualidad que identifica a la hoja de la planta de papa como sana. El segundo se refiere a la exactitud del proceso en la clasificación, ello quiere decir que todas las características extraídas por cada neurona poseen el error mínimo posible con respecto a C_1, en consecuencia, todos los patrones de salida se encuentran muy próximos al valor deseado.

La implementación del filtro espacial con elemento estructurante puede contribuir a mejorar la clasificación de imágenes altamente contaminadas. Se ha usado este tipo de filtro porque es el que menor influencia tiene sobre la morfología de la imagen con respecto a otros elementos estructurantes o técnicas de filtrado espacial, pues su degradación es mínima, ello no implica que todos los artefactos sean atenuados, pero que si sean aceptables al momento de clasificar la imagen. La Figura 5.3-3 muestra el resultado de la clasificación de una hoja sana mediante el uso de la herramienta construida.

Figura 5.3-3: Clasificación de una hoja sana



Nombre de la fuente: Elaboración propia

6. Resultados

Los resultados de este trabajo están reflejados en la organización de la base de datos, el comportamiento de las capas convolucionales, la dinámica de sus pesos, la exactitud en la clasificación y el comportamiento de la herramienta computacional.

6.1 Organización de la base de datos

Con el fin de obtener el entrenamiento de la red neuronal convolucional se han conformado dos conjuntos: uno de entrenamiento y otro de prueba. Se ha usado una distribución de imágenes en una relación 80 / 20, es decir; 80% para el primer conjunto y el 20% restante para el segundo. Esta distribución permitió que la ConvNet creada por el autor alcanzara una exactitud del 99.18% en su fase de validación, durante el proceso de entrenamiento. La base de datos está compuesta por varias carpetas y subcarpetas, la carpeta principal es llamada "*PlantVillage*", la cual contiene una subcarpeta llamada "*imágenes_de_hojas_de_papa*", que a su vez contiene dos subcarpetas nombradas "*papa_con_tizón_tardío*" y "*papa_sana*", estos dos últimos nombres conforman las etiquetas de clasificación que serán configuradas y usadas en la red.

Con base en esta organización se efectúa el contado de etiquetas en las últimas dos subcarpetas, como era de esperarse, la menor cantidad se atribuye a la carpeta de *papa_sana*, con 304 elementos, ellos son 152 imágenes crudas más 152 imágenes pre – procesados, así que el tamaño de cada conjunto está en función de la menor cantidad de elementos que se ha identificado los cuales son asignados de forma aleatoria. Luego se efectúa una relación de 80 / 20 con el fin de optimizar la carga computacional en el proceso de validación, asimismo, se ha elegido un tamaño de minilote equivalente a 38 elementos los cuales dividen el conjunto de entrenamiento en 8 minilotes, ello con el fin de optimizar el costo computacional del cálculo de los pesos en la fase de entrenamiento.

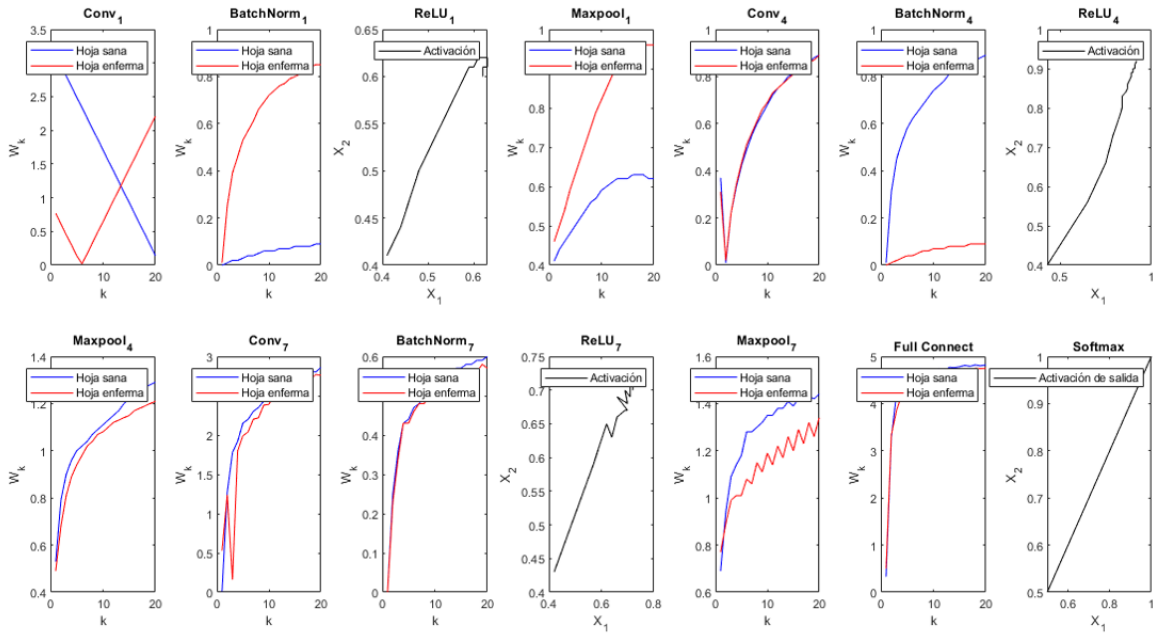
6.2 Comportamiento de las capas convolucionales

Usando la función Deep Dream pueden identificarse el comportamiento de cada una de las capas de la red neuronal, dicha función actúa de forma inversa profundizando en la CNN e identificando el comportamiento de las neuronas que la componen. A medida que la imagen va profundizando en la red los detalles de la misma van cambiando, por ejemplo, en la primera capa solo hay características básicas de la imagen que reflejan algunas componentes de color, así como una mezcla básica en los colores. Sin embargo, las características con alto nivel de resolución y una compleja mezcla de colores, enseñan información relevante de la imagen cuyas componentes son muy distintas una de la otra, ello puede apreciarse en las ultimas capas.

6.3 Dinámica de los pesos

Para obtener la dinámica de los pesos se han tomado dos imágenes; una de hoja sana y otra enferma. Ellas han sido pasadas 20 veces por la CNN con el fin de establecer el comportamiento de los pesos para cada una de sus capas, los resultados de esta prueba se muestran en la Figura 6.3-1:

Figura 6.3-1: Dinámica de los pesos sinápticos de la CNN



Nombre de la fuente: Elaboración propia

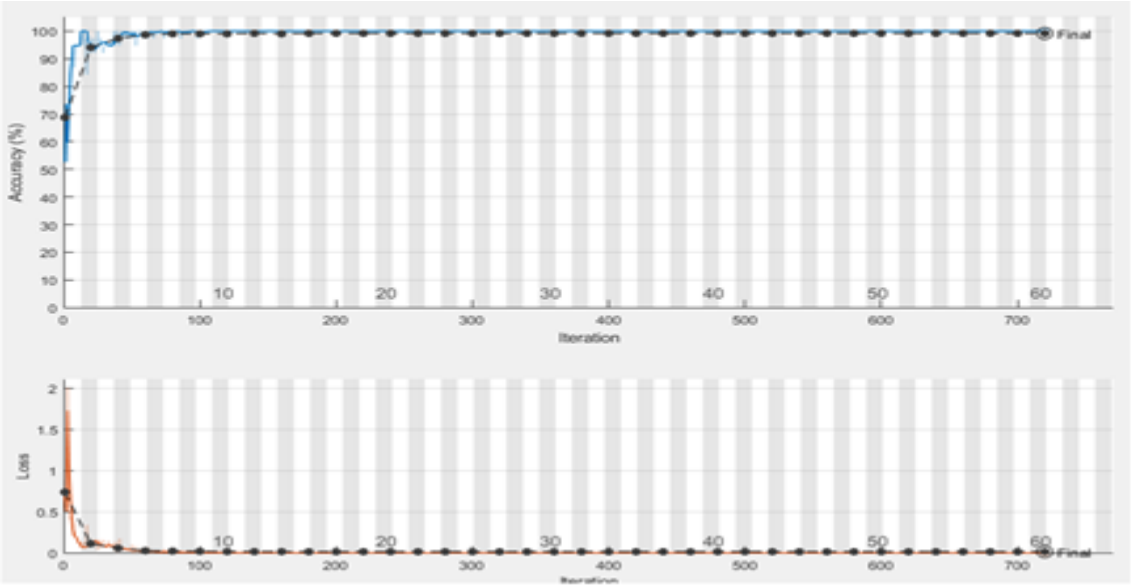
La dos entradas que ingresan a la red neuronal convolucional son designadas por x_1 y x_2, la Figura 6.3-1 muestra que la dinámica de $W(k)$ en el primer filtro posee un comportamiento lineal conforme el valor de la iteración k va en aumento, estas conexiones van ajustándose gracias a lo aprendido por la red, en la normalización de lote la dinámica de $W(k)$ es más evidente en x_2 que en x_1. La función de activación establece dos regiones de decisión delimitadas por una función de fase aproximadamente lineal, de aquí la función de agrupamiento extrae los pesos más representativos para x_1 y x_2. Análogamente, en la cuarta capa de convolución puede verse que las conexiones $W(k)$ son no lineales, siendo la normalización del lote más representativa para x_1, la activación ReLU va aproximándose a una función lineal, los pesos más significativos son clasificados gracias a la función *maxpooling*.

En la séptima capa ocurre un comportamiento no lineal, tal como se aprecia en la Figura 6.3-1, los pesos sinápticos en la capa completamente conectada se han ajustado de acuerdo a las características más representativas que se extrajeron en cada entrada y poseen el más alto nivel de fuerza sináptica, así la salida de la función *Softmax* establece dos regiones de decisión asociadas a C_1 y C_2. Lo anterior muestra que la CNN construida establece correctamente dichas regiones; la primera clase está ubicada en la parte inferior del hiperplano perteneciente al conjunto de hojas sanas siendo la parte superior del segundo conjunto (hojas enfermas), la línea recta establece la frontera de intersección de ambos conjuntos.

6.4 Métricas de la red

Durante el entrenamiento de la red se ha obtenido una exactitud del 99.18% para el proceso de validación, la red ha admitido 60 épocas de 100 programadas, cada una contiene 12 iteraciones para un total de 720 repeticiones, esto significa que cada lote pasó por la red 720 veces, donde la perdida de datos es inferior a la unidad. El resultado derivado del entrenamiento de la red es mostrado en la Figura 6.4-1:

Figura 6.4-1: Nivel de exactitud durante el entrenamiento de la red



Nombre de la fuente: Elaboración propia

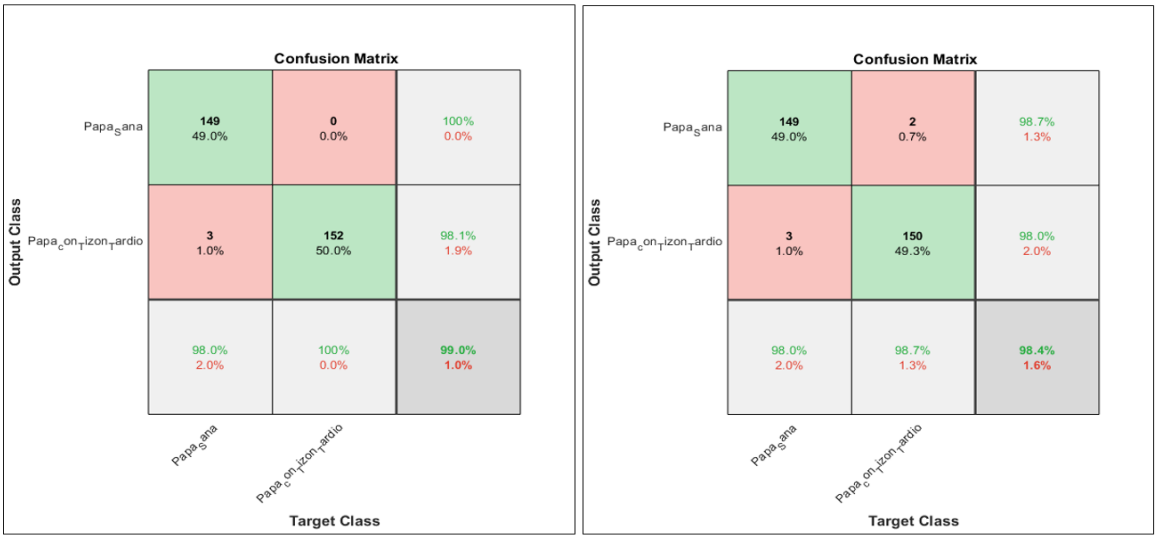
6.4.1 Matriz de confusión

Con el fin de evaluar la CNN mostrada en la cuarta sesión ha surgido la necesidad de aplicar las matrices de confusión, debido a la potencialidad que poseen al momento de valorar la clasificación de una ConvNet [32]. Para este proyecto, la CNN usa dos clases de clasificación que son C_1 y C_2 lo cual conlleva al origen de una matriz de confusión cuadrada de 2×2, misma que puede expresarse como $[MC_{11} \quad MC_{12}; MC_{21} \quad MC_{22}]$, donde MC_{11} se conoce como un verdadero positivo o TP por sus siglas en inglés, e indica el número de muestras cuya etiqueta actual es *Papa_Sana*. MC_{12} es un falso negativo o FN y muestra la cantidad de elementos mal clasificados con respecto a la etiqueta actual, MC_{21} denota la cantidad de elementos que fueron erróneamente clasificados como *Papa_Sana* y es un falso positivo FP. Por último, un verdadero negativo o TN ilustra las muestras que han sido clasificadas adecuadamente con la etiqueta *Papa_Con_Tizón_Tardio* y corresponde al elemento de la matriz MC_{22}.

En este trabajo la red neuronal construida por el autor se ha evaluado 15 veces, 5 con la base de datos PlantVillage con una cantidad de 304 imágenes, seguida de una base de datos de prueba conformada por 66 imágenes recolectadas de internet, finalmente, la evaluación ha sido llevada usando 22 imágenes la base de datos propia la misma cantidad

de veces que en los casos anteriores. En el algoritmo implementado en Matlab para estimar la matriz de confusión todas estas imágenes han sido asignadas como entradas a la CNN de forma aleatoria, obteniendo como resultado las matrices mostradas desde la Figura 6.4.1-1 hasta la Figura 6.4.1-8 para cada prueba de clasificación.

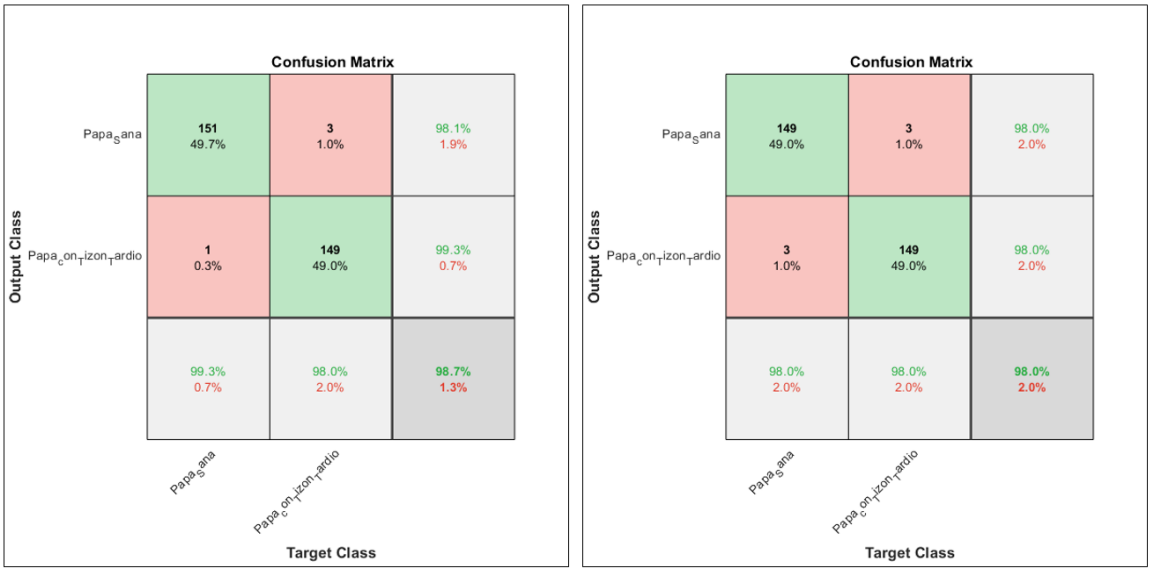
Figura 6.4.1-1: Matrices de confusión para las primera y segunda pruebas (PlantVillage)



Prueba PlantVillage -1 Prueba PlantVillage -2

Nombre de la fuente: Elaboración propia

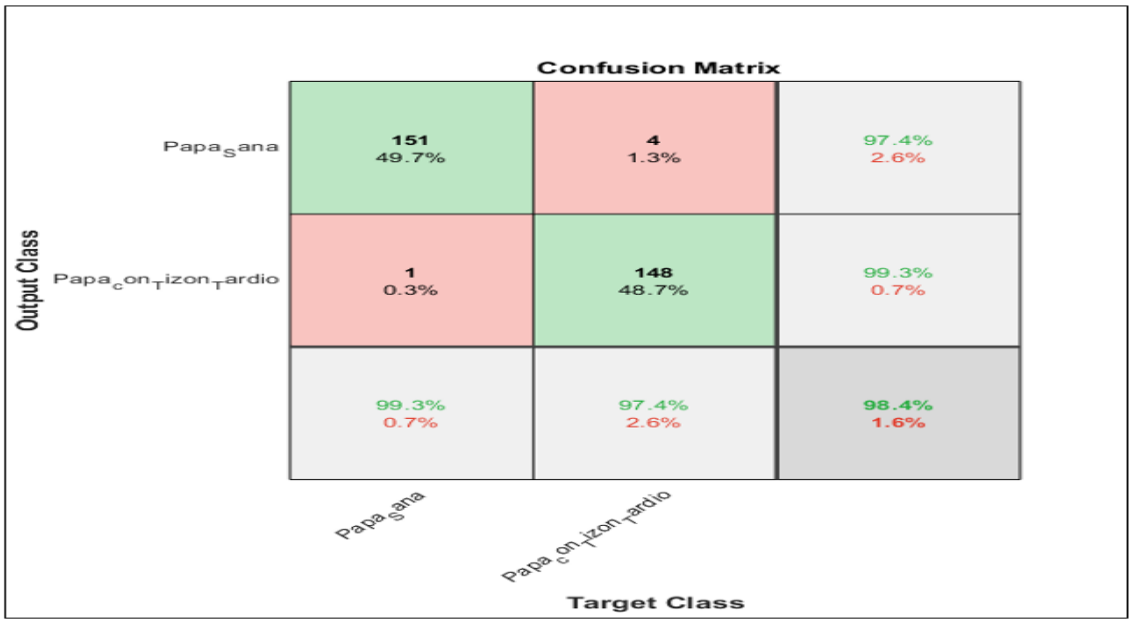
Figura 6.4.1-2: Matrices de confusión para las tercera y cuarta pruebas (PlantVillage)



Prueba PlantVillage - 3 Prueba PlantVillage - 4

Nombre de la fuente: Elaboración propia

Figura 6.4.1-3: Matriz de confusión para la quinta prueba (PlantVillage)



Prueba PlantVillage - 5

Nombre de la fuente: Elaboración propia

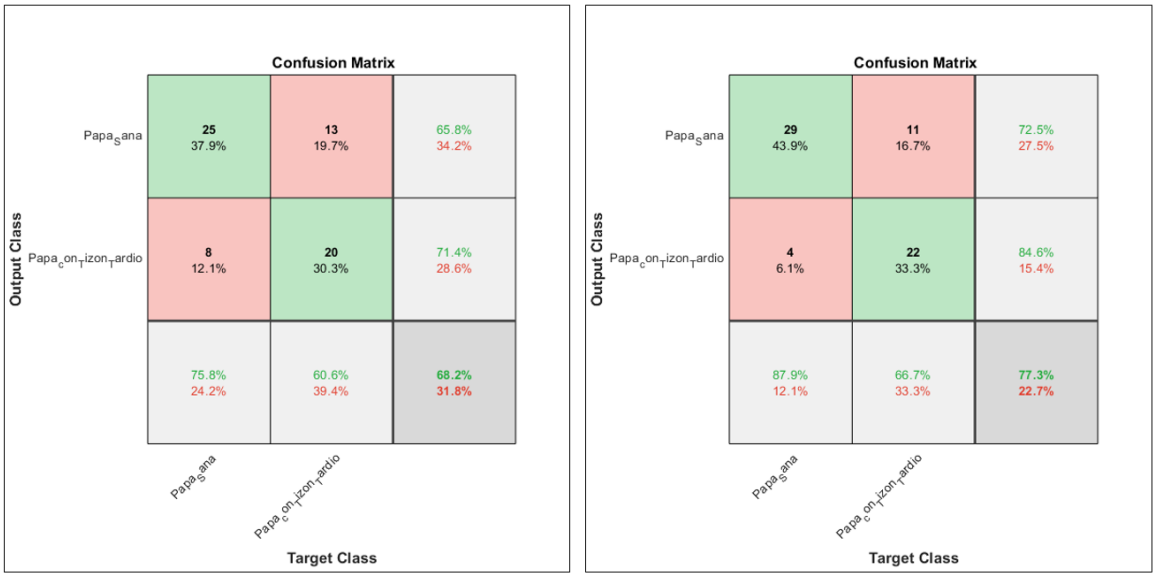
Figura 6.4.1-4: Matrices de confusión para las primera y segunda pruebas (Base de prueba)

Prueba - 1

Prueba - 2

Nombre de la fuente: Elaboración propia

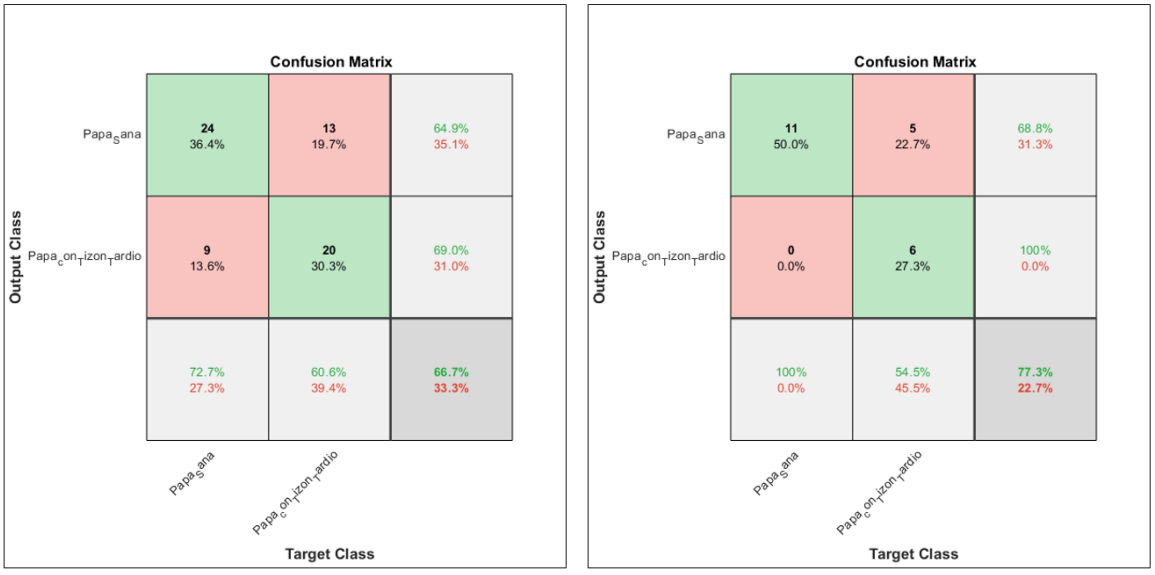
Figura 6.4.1-5: Matrices de confusión para las tercera y cuarta pruebas (B)



Prueba - 3 Prueba - 4

Nombre de la fuente: Elaboración propia

Figura 6.4.1-6: Matrices de confusión – quinta prueba y primera (Base de datos propia)



Prueba - 5 Propia - 1

Nombre de la fuente: Elaboración propia

Figura 6.4.1-7: Matrices de confusión para las segunda y tercera pruebas (Base de datos propia)

Propia - 2

Propia - 3

Nombre de la fuente: Elaboración propia

Figura 6.4.1-8: Matrices de confusión para las cuarta y quinta pruebas (Base de datos propia)

Propia - 4

Propia - 5

Nombre de la fuente: Elaboración propia

Los porcentajes contenidos en cada celda de la matriz indican el peso de la clasificación tanto correctas como erróneas, la última columna muestra el porcentaje de predicción bien

y mal clasificadas por la red en cada etiqueta, así como la última fila de la matriz. Esta columna indica la precisión mientras que la fila establece una métrica conocida como *recall*, la última celda de la matriz indica la exactitud total de la prueba realizada. Cada una de las métricas que puede estimarse por medio de la matriz de confusión son descritas en la siguiente sesión.

6.4.2 Cálculo de métricas en las pruebas de clasificación

Las métricas usadas para mostrar el comportamiento de la red durante las 15 pruebas realizadas son la exactitud, la precisión, el recall y el F1 – Score. La exactitud traduce la cercanía al valor deseado en la clasificación teniendo en cuenta el error mínimo posible y se representa matemáticamente como:

$$Exactitud = (TP + TN)/(TP + TN + FP + FN) \tag{6.1}$$

La precisión indica la cercanía de las muestras en cada etiqueta de clasificación teniendo en cuenta la cantidad de falsos positivos, computando la fracción de predicción de positivos y puede escribirse como sigue:

$$Precisión = TP/(TP + FP) \tag{6.2}$$

La métrica recall calcula la fracción de positivos actuales y es dada por:

$$Recall = TP/(TP + FN) \tag{6.3}$$

Por último, el F1 – Score es la media harmónica de la precisión y el recall, sirve para unificar estas dos métricas permitiendo al diseñador de redes neuronales obtener una medición más fiable en términos del rendimiento de la red, matemáticamente es expresada como sigue:

$$F1 = 2TP/(2TP + FP + FN) \tag{6.4}$$

Los resultados obtenidos a aplicar las ecuaciones anteriormente mencionadas se observan en las siguientes tablas:

Tabla 6-1: Resultado de las pruebas con la base de datos PlantVillage.

Prueba	Base de datos	Cantidad de imágenes usadas	Exactitud	Precisión	Recall	F1 – Score
1	PlantVillage	304	99.01	100	98.03	99.0
2	PlantVillage	304	98.36	98.68	98.03	98.35
3	PlantVillage	304	98.68	98.05	99.34	98.69
4	PlantVillage	304	98.03	98.03	98.03	98.03
5	PlantVillage	304	98.36	97.42	99.34	98.37

Tabla 6-2: Resultado de las pruebas con la base de datos de prueba.

Prueba	Base de datos	Cantidad de imágenes usadas	Exactitud	Precisión	Recall	F1 - Score
1	Prueba	66	77.27	76.47	78.79	77.61
2	Prueba	66	71.21	66.67	84.85	74.67
3	Prueba	66	68.18	65.79	75.76	7042
4	Prueba	66	77.27	72.50	87.88	79.45
5	Prueba	66	66.67	64.86	72.73	68.57

Tabla 6-3: Resultado de las pruebas con la base de datos de Propia.

Prueba	Base de datos	Cantidad de imágenes usadas	Exactitud	Precisión	Recall	F1 - Score
1	Propia	23	77.27	68.75	100	81.48
2	Propia	23	68.18	62.50	90.91	74.07
3	Propia	23	63.64	58.82	90.91	71.43
4	Propia	23	77.27	68.75	100	81.84
5	Propia	23	77.27	68.75	100	81.84

Lo anterior muestra que el mayor desempeño de la red fue en la clasificación de imágenes de la base de datos PlantVillage con un nivel máximo del 99% y 98.03% como valor mínimo, puede afirmarse que este desempeño fue moderado al observar la Tabla 6-3, dado que el nivel máximo de F1 da lugar en la cuarta y quinta prueba con la base de datos propia, ambos con una métrica de 81.84% siendo el mínimo en 71.43% atribuido a la tercera prueba. Los peores resultados son mostrados en la Tabla 6-2 con un máximo de 79.45% en la cuarta prueba y un mínimo de 68.57% en la quinta, para la métrica F1 – Score.

6.5 Comportamiento de la herramienta

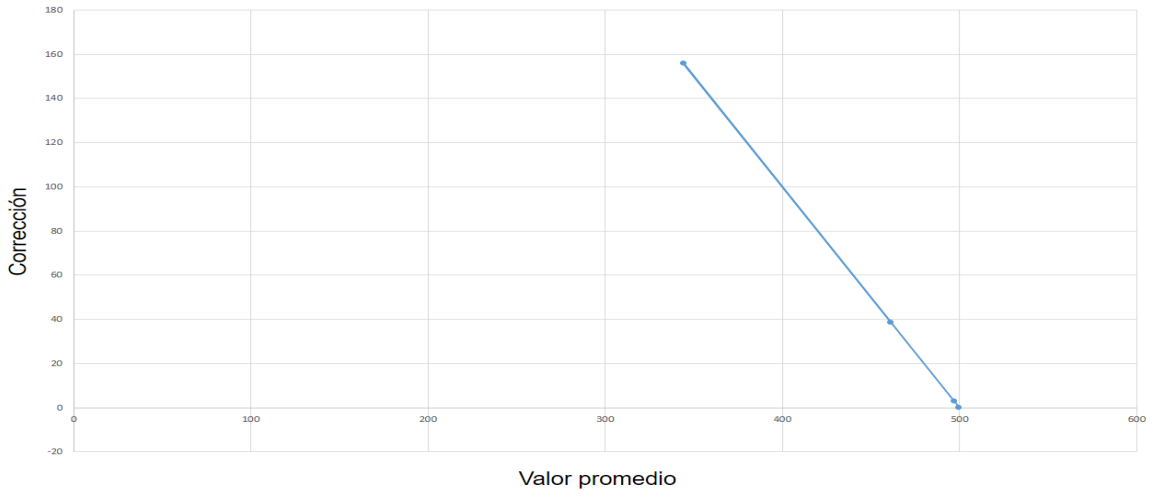
La herramienta computacional ha sido sometida a varias pruebas de repetitividad con el fin de observar la desviación que ella posee al momento de clasificar las imágenes de prueba, dichas imágenes están preparadas como hojas sanas y enfermas, los resultados obtenidos han sido mostrados en la Tabla 6-4:

Tabla 6-4: Comportamiento de la herramienta con imágenes de prueba

Cantidad de imágenes	Tipo de imagen a clasificar	Cantidad promedio clasificadas	Error	U_e	Resultado de la prueba
500	Hoja sana	461.6	-38.4	2.417	No aprueba
500	Hoja sana con filtrado	345	-155	2.367	No aprueba
500	Hoja enferma	497.2	-2.8	2.400	Aprueba
500	Hoja enferma con filtrado	500	0	2.367	Aprueba

La curva que caracteriza el comportamiento de la herramienta puede verse en la Figura 6.5-1:

Figura 6.5-1: Curva de comportamiento de la herramienta computacional



Nombre de la fuente: Elaboración propia

Los datos expresados en la Tabla 6-4 han sido calculados basándose en la guía para estimar la incertidumbre de medida hallada en [48], ello se basa en métricas estadísticas para evaluar si el resultado de una medición posee un alto nivel de confiabilidad o no.

Con base en la prueba a la que ha sido sometida la herramienta, la repetitividad consiste en determinar la capacidad de identificar las imágenes sanas y enfermas al ingresar 500 imágenes de PlantVillage, se toman como elementos patrón, dado que de antemano se conocen como hojas sanas y de tizón tardío respectivamente, razón por la cual se han usado para entrenar la red neuronal. En condiciones ideales, la *NeuroPI – 2105* debería ser capaz de clasificar todas las imágenes, por lo cual se establece manualmente un conteo en la clasificación por cada imagen que se ingrese como entrada a la herramienta. De este modo, se explica por ejemplo que ante la repetición de 5 veces la predicción de 500 imágenes se han contado un promedio de 497.2 imágenes bien clasificadas con un error de 2.8 imágenes clasificadas de forma errónea, el signo indica que el error se mueve en sentido anti horario con respecto a la referencia que son 500 imágenes.

La prueba ha mostrado que la herramienta ofrece mayor nivel de confianza cuando las clasificaciones de las imágenes corresponden a hojas de plantas de papa con tizón tardío, el menor error fue obtenido aplicando la función “Filtro”, aquí también se atribuye la menor incertidumbre expandida U_e durante la prueba, la cual ha sido estimada usando un factor de cubrimiento $k = 2$. Desgraciadamente, la *NeuroPI-2105* presenta los más altos niveles de error en la prueba de clasificación de hojas sanas, aun así, se ha logrado identificar adecuadamente las hojas enfermas de tizón tardío usando como referencias 500 imágenes de estas hojas, mismas que fueron tomadas de PlantVillage. Por último, la herramienta se ha sometido a prueba con 108 imágenes atribuidas a hojas enfermas de tizón tardío recolectadas desde internet y de la base de datos propia con resultados satisfactorios, pues solo dos de ellas fueron clasificadas de forma errónea.

Por último, no se entrará en detalles para expresar los términos de error, corrección, U_e y factor de cubrimiento dado que esta fuera del alcance de este proyecto. Sin embargo, se sugiere al lector interesado consultar [48, p. 18] para un mayor entendimiento en el cálculo de la incertidumbre.

7. Conclusiones

Se ha obtenido una adecuada organización en la base de datos la cual permitió entrenar la red neuronal convolucional con una exactitud del 99.18% en la fase de validación. A su vez dicha base de datos fue aplicada en las pruebas de funcionalidad de la herramienta alcanzando resultados favorables en la identificación de hojas enfermas de tizón tardío. Sin embargo, al desarrollar numerosas pruebas con las bases de datos relacionadas en la Tabla 3-1, se ha concluido que cada una de las imágenes allí descritas son necesarias más no suficientes, por el hecho de que en los cultivos colombianos las cualidades de las imágenes difieren mucho de las halladas en la base de datos usadas. Ello es debido a la naturaleza con las que ellas son adquiridas, por lo cual se sugiere ahondar en los mecanismos hallados en la literatura para su pre – procesamiento, aunque ello incrementa sin duda el costo computacional.

Teniendo en cuenta la base de datos usada para este proyecto, la implementación de la red convolucional contempló tres etapas que son: el diseño, la arquitectura y el entrenamiento, al emplear cada una de ellas, se ha producido una CNN con 26.631 parámetros, siete capas de convolución y una exactitud del 99.18%. La red es capaz de clasificar imágenes de hojas sanas y con tizón tardío, se ha sometido a varias pruebas de repetitividad notando que su comportamiento posee mayor capacidad al momento de clasificar imágenes de hojas enfermas. Su desarrollo e implementación se efectuó en Matlab 2018 con resultados satisfactorios, existiendo dinámica en la actualización de los pesos, así como los bias, la salida de la red segmenta adecuadamente las dos clases de clasificación, por lo cual se concluye que esta red cuenta con las cualidades suficientes para identificar hojas enfermas con tizón tardío.

La herramienta computacional fue diseñada a partir de los materiales y métodos relacionados en la Tabla 4-1, llevando a la interfaz mostrada en la Figura 5.3-1. Dicha

herramienta es nombrada *NeuroPI - 2105* tal como se explica en la sesión 5.3, su operatividad es concebida gracias al mapa de navegación ilustrado en la Figura 5.1-2. Se ha obtenido un producto computacional fácil de usar, es portable y ejecutable en cualquier sistema operativo Windows para computadoras portátiles y de escritorio, no necesita el empleo de Matlab para su operación. Las indicaciones de texto orientan al usuario en el manejo de la misma, se ha pensado en una interfaz simple y fácil de usar, que permite cargar o cancelar las imágenes, a su vez, posee un algoritmo que identifica si la imagen es válida o no por medio del SSIM, integra la opción de un filtro espacial para atenuar artefactos, mismos que pueden ser visualizados por medio del botón *Artefactos*.

Su construcción es empaquetada en una carpeta que incluyen archivos de extensión ".exe", ello permite su uso sin la dependencia de Matlab para su funcionamiento. Las pruebas de repetitividad en la clasificación de imágenes arrojaron un error pronunciado de -38.4 en la clasificación de imágenes asociadas a hojas sanas sin tratamiento y de -155 aplicando a las mismas el filtro espacial, por tal motivo no se sugiere su uso para estas tareas. Para el caso de las hojas enfermas y a pesar del resultado anterior, la misma prueba con 500 imágenes de tizón tardío fue aplicada, con resultados favorables, ello considerando que el error máximo permitido (tolerancia) para la herramienta es de 4 imágenes mal clasificadas.

Se concluye que el software creado aporta a la necesidad de clasificar hojas enfermas de tizón tardío ante el problema estudiado en el primer capítulo del proyecto, la corrección promedio debe ser de 2.8 imágenes para imágenes no procesadas, pese a ello, al aplicar el filtro pasa bajas todas ellas son clasificadas, presentando un error nulo en dicha actividad, es decir, la medida está dentro del intervalo $[497.633, 502.367]$ proporcionando un nivel de confianza del 95.4%, al tomar como referencia 500 imágenes de gota. Por lo tanto, se ha determinado que la herramienta identifica dos clases de hojas de herbáceos de papa que son: hojas sanas y enfermas de tizón tardío. Aunque es capaz de clasificar ambas clases de hojas, se ha comprobado que tiende a aumentar el error en la clasificación de hojas sanas en comparación con las enfermas, sugiriendo su uso para identificar la segunda clase de hojas, también se recomienda aplicar la acción de filtro antes de clasificarlas.

Aunque el aumento pronunciado del error dado en la prueba de repetitividad descrita en la sesión 6.5 para la primera clase de hojas conlleva a una preocupación del autor concerniente a su aplicabilidad en el sector agricultor, es evidente que dicha prueba describe una adecuada clasificación de hojas enfermas, ello también ha sido verificado a través de la base de datos propia por lo cual se concluye que el software clasifica imágenes de hojas asociadas a la enfermedad de tizón tardío en cultivos de papa.

7.1 Recomendaciones

En la ejecución de este proyecto se han presentado algunos aspectos que han conllevado a plantear importantes desafíos e interrogantes para el autor que sin duda deben ser estudiados, centrándose en cuatro retos que son; la conformación de una sólida base de datos, el tratamiento digital de las imágenes, la elección de la arquitectura de la red neuronal y la calidad de experiencia que ofrece la herramienta computacional a los usuarios, es decir, a los campesinos.

Si bien la base de datos PlantVillage es necesaria, ella no es suficiente. Los cultivos de papa colombianos presentan diversidad de pestes en su flora, se ha evidenciado que el agricultor se basa en su experiencia para identificar la enfermedad del tizón tardío, ello puede causar que él tome decisiones que puedan afectar la productividad de la siembra debido a un mal diagnóstico o uso inadecuado en los productos para combatir la enfermedad. Este escenario plantea la necesidad de construir un repositorio de imágenes científicamente avalado, que sirva como "*imágenes patrón*" para entrenar la CNN y que permita evaluar su fiabilidad para implementarla en herramientas de diagnóstico.

Al elegir una técnica para el tratamiento de imágenes, se sugiere primero establecer un modelo matemático que permita describir el comportamiento de esta enfermedad, ello puede involucrar cierto nivel de profundidad del conocimiento en ciencias biológicas. Sin embargo, una buena aproximación del modelo podrá ayudar a plantear la morfología que produce las lesiones del tizón tardío en las hojas de papa y, por tanto, una buena técnica para el tratamiento de este tipo de imágenes.

En cuanto a la arquitectura de la CNN, se podría pensar en construir una red de muchas capas, lo cual causa un elevado costo computacional, por lo tanto, se advierte la necesidad

de estudiar a profundidad las arquitecturas de redes como AlexNet, ResNet, Inception, entre otras, para minimizar o eliminar este problema.

Ante lo anterior se sugieren las siguientes recomendaciones para futuros aportes a este proyecto:

- Construir una base de datos a partir de cultivos colombianos la cual sea llevada a cabo por un equipo interdisciplinar en áreas de la bioingeniería, ingeniería agrónoma, ingeniería electrónica y la biología, como mínimo.
- Implementar una función de segmentación morfológica que surja del modelo matemático que describa el comportamiento de la enfermedad.
- Diseñar y construir una red que permita minimizar al máximo el costo computacional al momento de ser entrenada o re – entrenada.
- Mejorar la interactividad y calidad de experiencia de la herramienta elaborada en este trabajo, para ello debe someterse a evaluación para detectar posibles fallas, errores y demás necesidades del usuario final, los cuales deben ser mejorados. La herramienta debe someterse a juicio del agricultor, así como de expertos hasta alcanzar el objetivo que se haya planteado el investigador.

A. Anexo: Construcción de la red neuronal convolucional

El algoritmo usado para construir la ConvNet empleada en este proyecto es como el mostrado a continuación:

Algoritmo A1: Construcción de la red neuronal convolucional

```
% CONSTRUCCION DE LA RED NEUTONAL CONVOLUCIONAL
% Dataset: PlantVillage
% Arquitectura 2-32-64-64-128-128-256-256-2
% Elaborada por: Camilo Andrés Ortiz Daza
% Universidad Antonio Nariño - Bogotá

CarpetadeSalida = fullfile('Plantvillage');
CarpetaRaiz = fullfile(CarpetadeSalida,'Imagenes_de_Hojas_de_Papa');
Categorias = {'Papa_con_Tizon_Tardio','Papa_Sana'};
imds =
imageDatastore(fullfile(CarpetaRaiz,Categorias),'LabelSource','foldernam
es');
tabla = countEachLabel(imds);
minimoConjuntoConteo = min(tabla{:,2});
imds = splitEachLabel(imds, minimoConjuntoConteo, 'randomize');
[ConjuntoEntrenamiento, ConjuntoPrueba] = splitEachLabel(imds, 0.8,
'randomize');
%ARQUITECTURA DE LA RED NEURONAL CONVOLUCIONAL
Capas = [ ...
    imageInputLayer([256 256 3])
    % Capa 1
    convolution2dLayer(5,32,'Stride',1,'Padding',2)
```

```
batchNormalizationLayer
reluLayer
maxPooling2dLayer(2,'Stride',2)
% Capa 2
convolution2dLayer(5,64,'Stride',1,'Padding',2)
batchNormalizationLayer
reluLayer
maxPooling2dLayer(2,'Stride',2)
% Capa 3
convolution2dLayer(5,64,'Stride',1,'Padding',2)
batchNormalizationLayer
reluLayer
maxPooling2dLayer(2,'Stride',2)
% Capa 4
convolution2dLayer(5,128,'Stride',1,'Padding',2)
batchNormalizationLayer
reluLayer
maxPooling2dLayer(2,'Stride',2)
% Capa 5
convolution2dLayer(5,128,'Stride',1,'Padding',2)
batchNormalizationLayer
reluLayer
maxPooling2dLayer(2,'Stride',2)
% Capa 6
convolution2dLayer(5,256,'Stride',1,'Padding',2)
batchNormalizationLayer
reluLayer
maxPooling2dLayer(2,'Stride',2)
% Capa 7
convolution2dLayer(5,256,'Stride',1,'Padding',2)
batchNormalizationLayer
reluLayer
dropoutLayer
maxPooling2dLayer(2,'Stride',2)
fullyConnectedLayer(2)
softmaxLayer
classificationLayer('Name','Salida')];
```

```
entrenamientoRed = trainingOptions('adam',...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropFactor',0.2,...
    'LearnRateDropPeriod',5,...
    'MaxEpochs',50,...
    'MiniBatchSize',38,...
    'ValidationData',imds,...
    'ValidationFrequency',20,...
    'Plots','training-progress');
Red = trainNetwork(ConjuntoEntrenamiento,Capas,entrenamientoRed);
RedCNNHojasPapa = Red;
save RedCNNHojasPapa1
save RedCNNHojasPapa1.data
```

B. Anexo: Construcción de la herramienta computacional

El algoritmo que permitió crear el software para el diagnóstico de hojas con tizón tardío es mostrado de la siguiente forma:

Algoritmo B1: Construcción de la herramienta computacional

```
function varargout = ConvNETToolPapa(varargin)
% CONVNETTOOLPAPA MATLAB code for ConvNETToolPapa.fig
%      CONVNETTOOLPAPA, by itself, creates a new CONVNETTOOLPAPA or
raises the existing
%      singleton*.
%
%      H = CONVNETTOOLPAPA returns the handle to a new CONVNETTOOLPAPA
or the handle to
%      the existing singleton*.
%
%      CONVNETTOOLPAPA('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in CONVNETTOOLPAPA.M with the given input
arguments.
%
%      CONVNETTOOLPAPA('Property','Value',...) creates a new
CONVNETTOOLPAPA or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before ConvNETToolPapa_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to ConvNETToolPapa_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ConvNETToolPapa
```

```
% Last Modified by GUIDE v2.5 23-Apr-2021 22:35:30

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ConvNETToolPapa_OpeningFcn, ...
                   'gui_OutputFcn',  @ConvNETToolPapa_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

global I
global R


% End initialization code - DO NOT EDIT


% --- Executes just before ConvNETToolPapa is made visible.
function ConvNETToolPapa_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ConvNETToolPapa (see VARARGIN)

% Choose default command line output for ConvNETToolPapa
handles.output = hObject;
load RedCNNHojasPapaf
R = Red;
handles.R = R;
axes1 = gca;
axes1.XAxis.Visible = 'off'; % Remueve las axisas del eje X
axes1.YAxis.Visible = 'off';% Remueve las axisas del eje Y

% Update handles structure
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.


% UIWAIT makes ConvNETToolPapa wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = ConvNETToolPapa_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.output = hObject;
[fn pn] = uigetfile('*.jpg','select jpg file');
if fn == 0
    set(handles.edit1,'string','Imagen no cargada');
    return
end
str = strcat(pn,fn);
%set(handles.edit1,'string',str); % mostrar el texto en la primera caja
de texto Tag = edit1
I = imread(str);
I = imresize(I, [256 256],'nearest');
handles.I = I;
imshow(I, 'parent',handles.axes1);
set(handles.edit1,'string','Presiona Aceptar');
guidata(hObject, handles);


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles = guidata(hObject);
I = handles.I;
H = fspecial('disk',1);
I = imfilter(I,H,"replicate");
imshow(I)
handles.I1 = I;
set(handles.edit1,'string','Presiona Clasificar');

% set(hObject,'value',5);
guidata(hObject, handles);


% --- Executes on button press in pushbutton3.
```

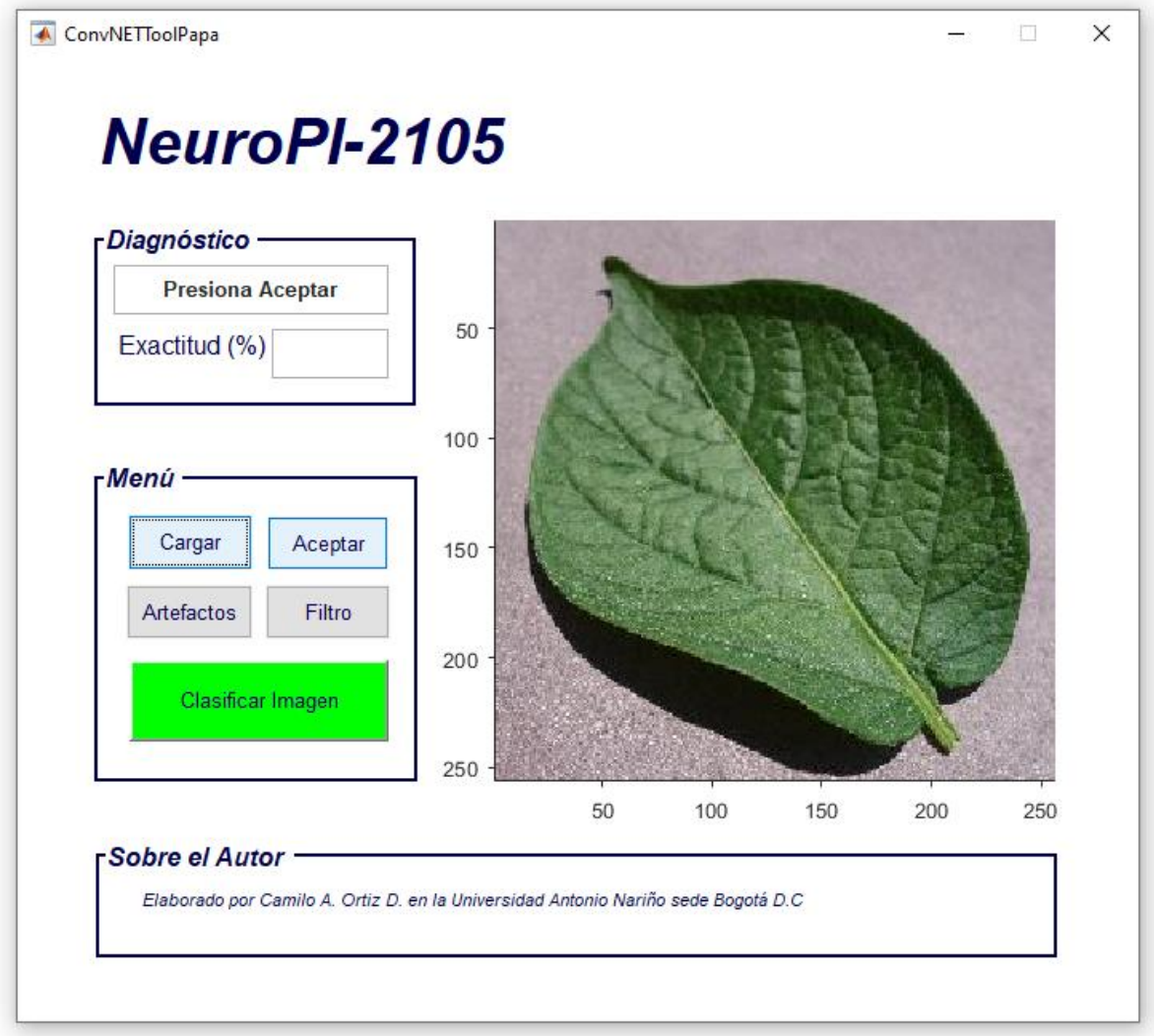
```
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles = guidata(hObject);
I = handles.I;
H = fspecial('disk',2);
%
% I = I-imfilter(I,H,"replicate");
gsI = rgb2gray(I);
BWI =
imbinarize(gsI,'adaptive','ForegroundPolarity','dark','Sensitivity',0.5)
;
SE = fspecial('disk',1);
gssmooth = imfilter(gsI,SE,"replicate");
BWfiltrada =
imbinarize(gssmooth,'adaptive','ForegroundPolarity','dark','Sensitivity'
,0.5);
ArtefactosImagen = imbinarize((BWI -
BWfiltrada),'adaptive','ForegroundPolarity','dark','Sensitivity',0.5);
imshow(ArtefactosImagen)
set(handles.edit1,'string','Artefactos-Presiona Aceptar');
guidata(hObject, handles);

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%
handles = guidata(hObject);
I = handles.I;
handles.I1 = I;
imshow(I)
set(handles.edit1,'string','Presiona Clasificar');
guidata(hObject, handles);

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles = guidata(hObject);
R = handles.R;
I = handles.I1;
imshow(I)
Iref = imresize(imread('Referencia.jpg'),[256 256]);
ssimBP = (ssim(I,Iref))*100
set(handles.edit1,'string','Presione Aceptar');
if ssimBP < 10
    set(handles.edit1,'string','Imagen no valida'); % mostrar el texto
en la primera caja de texto Tag = edit
    set(handles.edit2,'string',''); % mostrar el texto en la primera
caja de texto Tag = edit1
else ssimBP >= 10
```

```
    set(handles.edit1,'string','Imagen cargada'); % mostrar el texto en
la primera caja de texto Tag = edit1
    [pred,scores]=classify(R,I);
    Exactitud = max(double(scores*100));
    if pred == 'Papa_Sana'
        set(handles.edit1,'string','Hoja de papa sana');
        set(handles.edit2,'string',Exactitud);
    else pred == 'Papa_con_Tizon_Tardio';
        set(handles.edit1,'string','Hoja de tizón tardió');
        set(handles.edit2,'string',Exactitud);
    end
end




function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as
a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as
a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

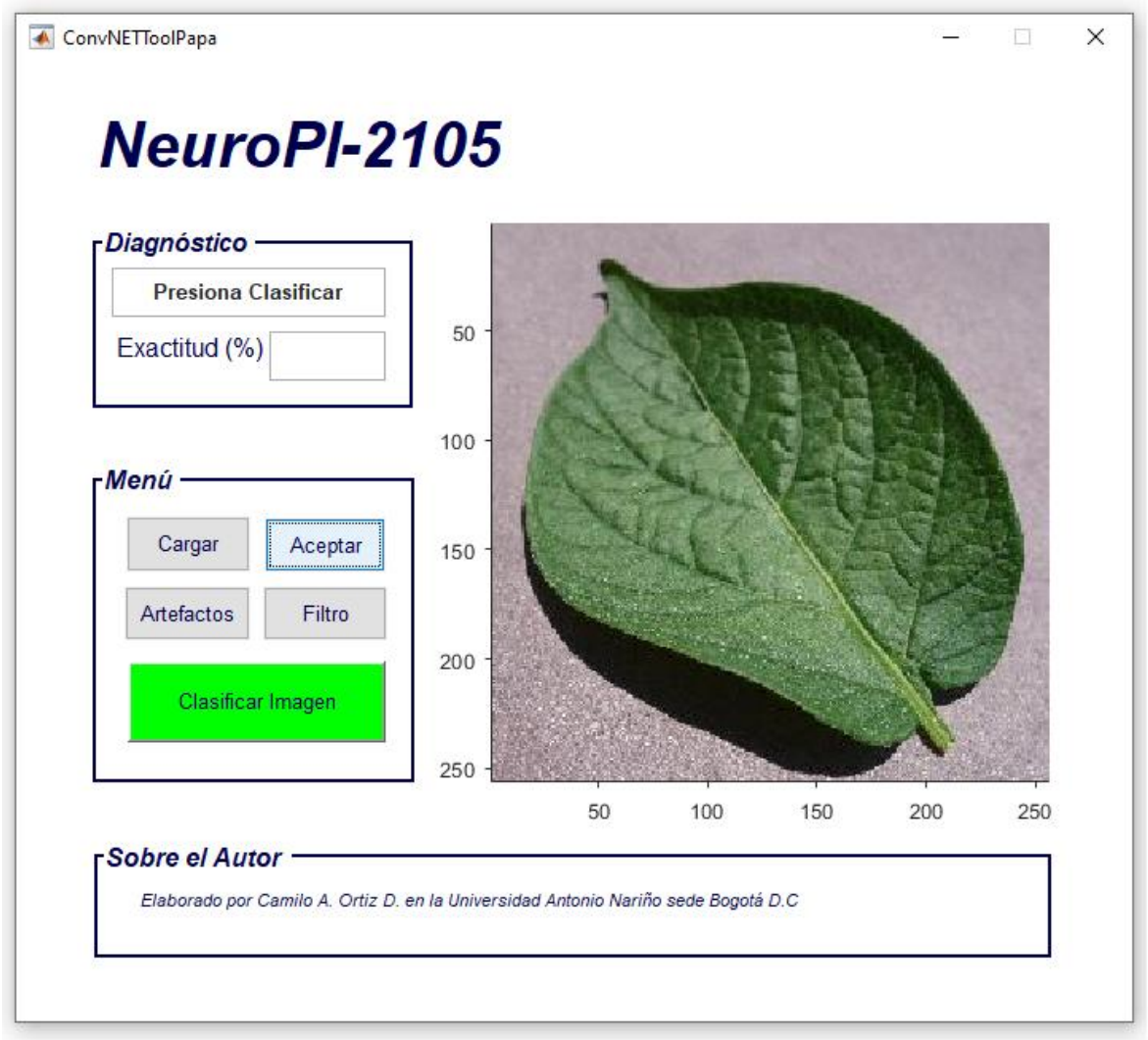
La apariencia de la herramienta puede ser apreciada en las siguientes figuras:

Figura B.1-1: Imagen a evaluar cargada



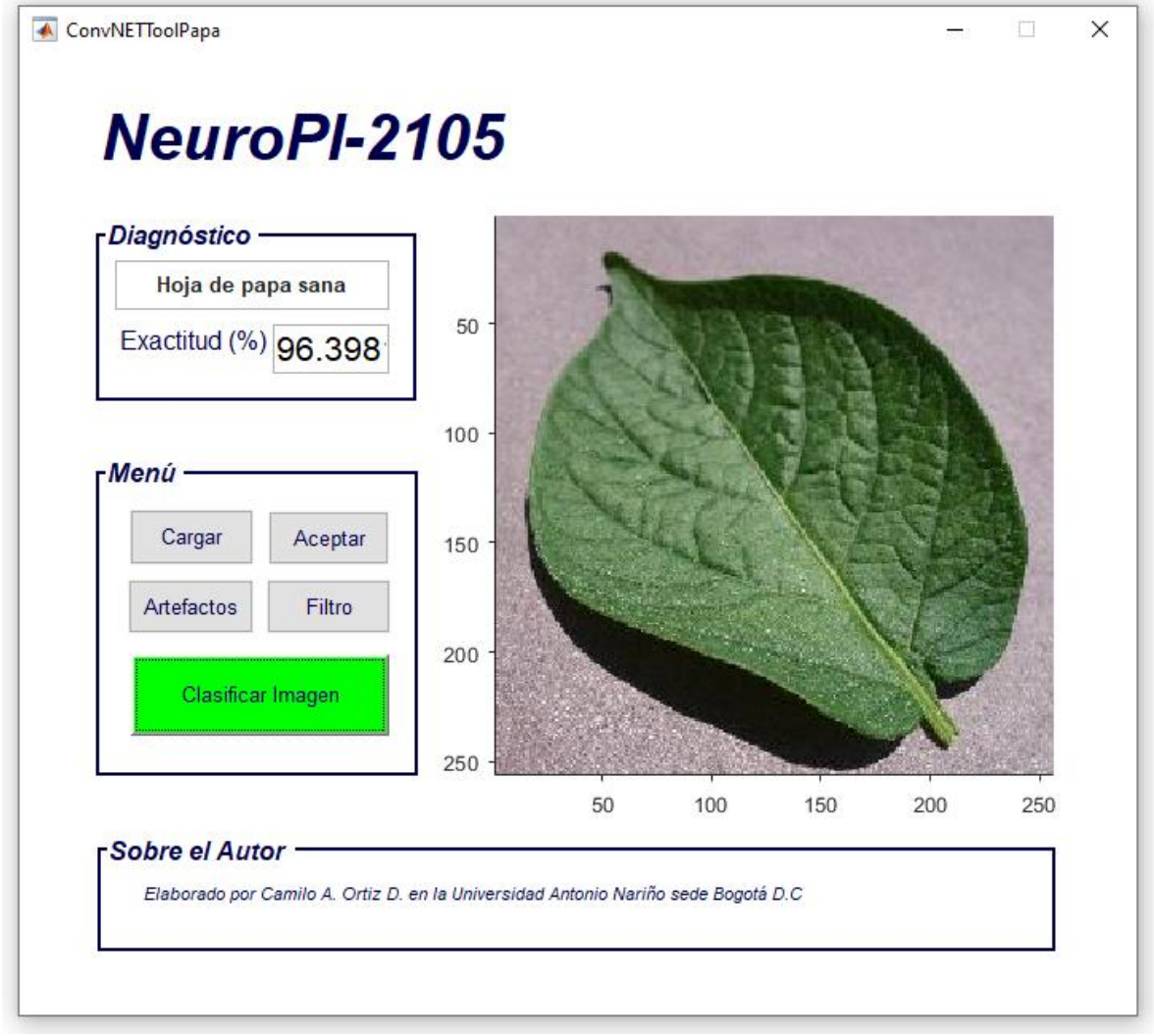
Nombre de la fuente: Elaboración propia

Figura B.1-2: Acción del botón aceptar



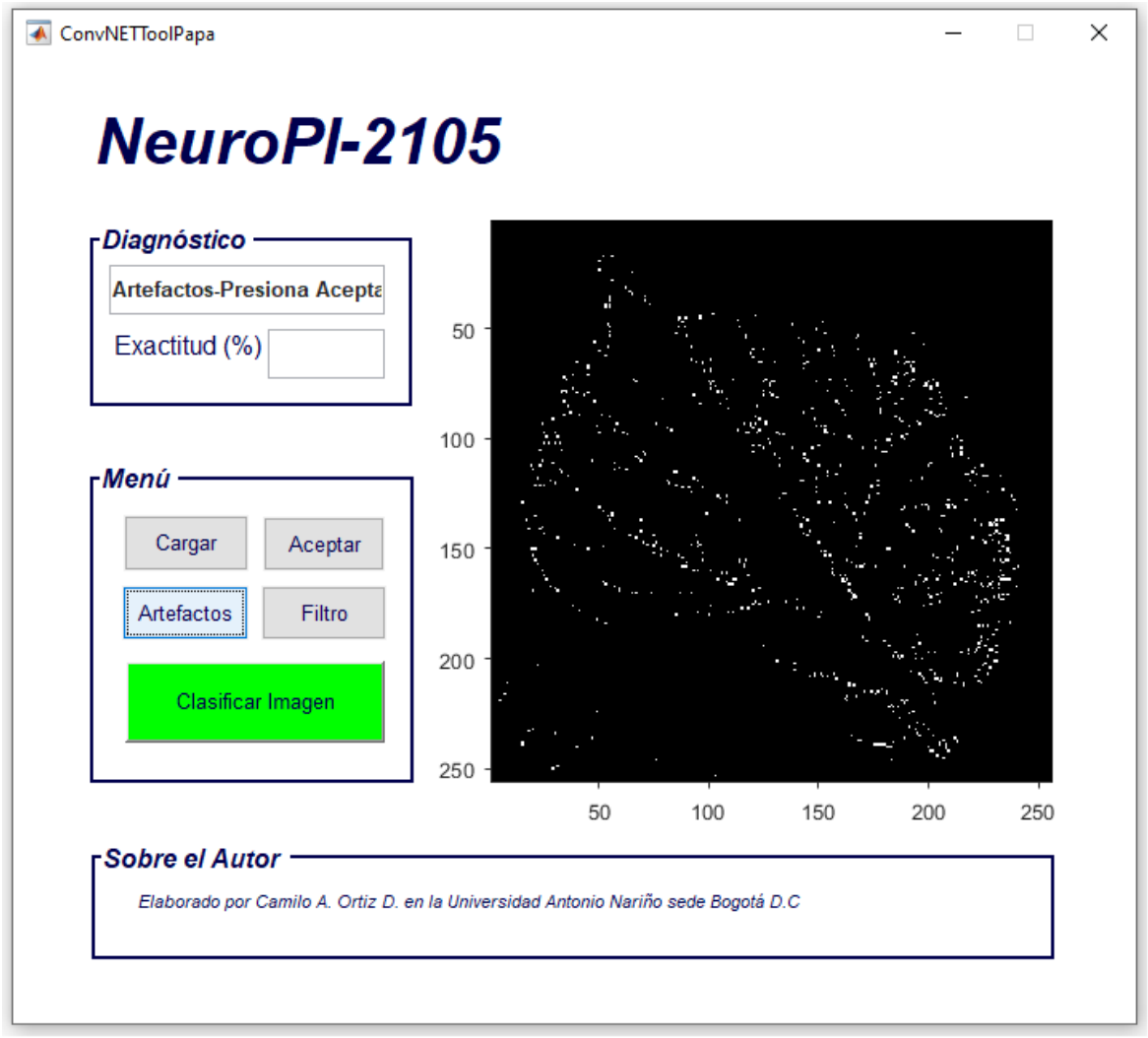
Nombre de la fuente: Elaboración propia

Figura B.1-3: Muestra del resultado al presionar el botón Clasificar



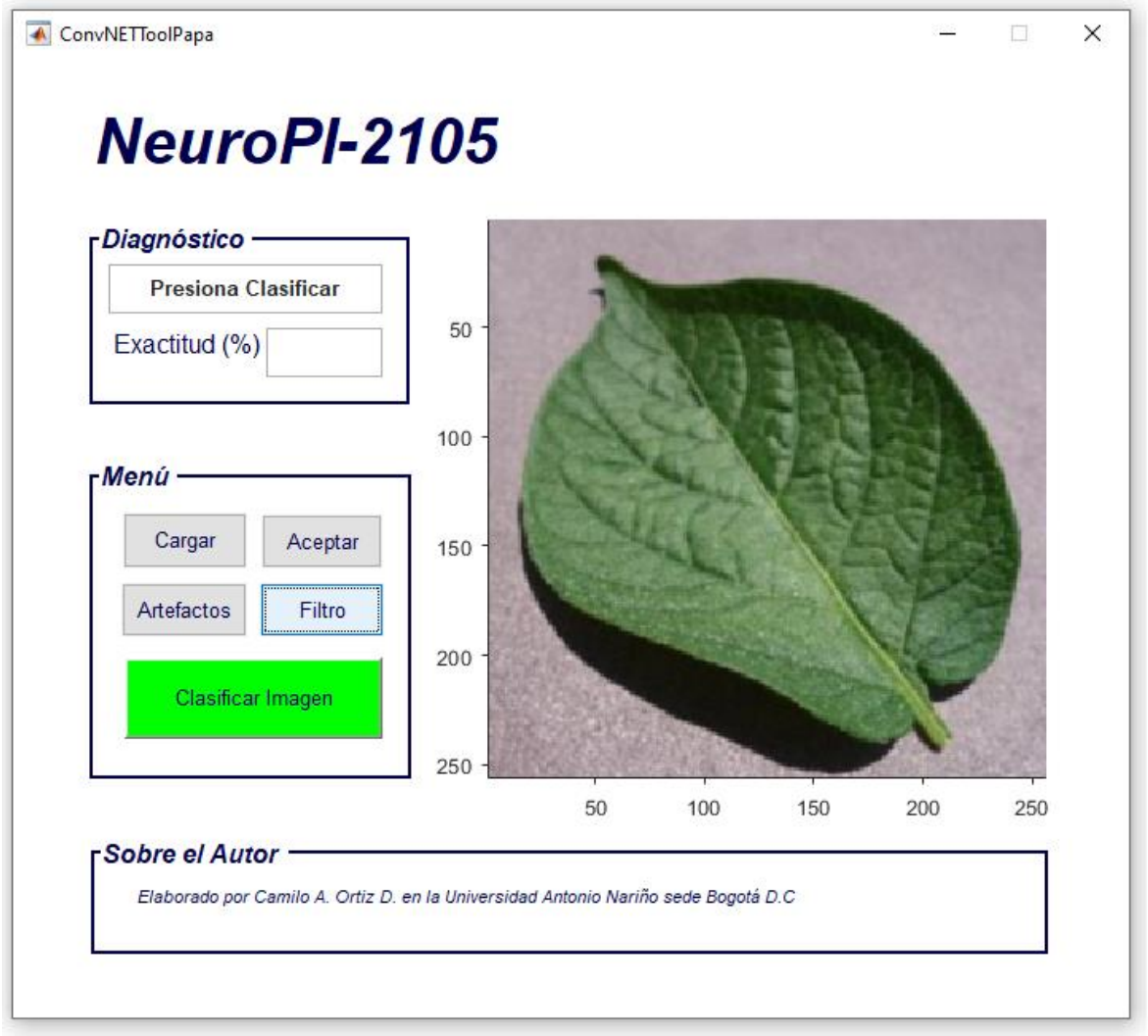
Nombre de la fuente: Elaboración propia

Figura B.1-4: Artefactos de la imagen



Nombre de la fuente: Elaboración propia

Figura B.1-5: Imagen filtrada



Nombre de la fuente: Elaboración propia

C. Anexo: Pre – procesamiento de las imágenes

El presente algoritmo muestra la etapa de procesamiento de que puede implementarse en los datos de *PlantVllage*:

Algoritmo C1: Etapa de pre – procesamiento

```
% ETAPA DE PREPROCESAMIENTO
% IMAGEN DE ENTRADA + SEGMENTACION + BINARIZACION + FILTRO + IMAGEN DE
% SALIDA

% Elaborada por: Camilo Andrés Ortiz Daza
% Universidad Antonio Nariño - Bogotá


clc
clear
baseImagenes = imageDatastore(fullfile('ImagenesPruebaGota'));
for k = 1:25
f = readimage(baseImagenes,k);
f = imresize(f,[256 256]);
figure(1)
imshow(f)
gsf = rgb2gray(f);
figure(2)
subplot(5,5,k)
imshowpair(f,gsf,'montage')
BWf =
imbinarize(gsf,'adaptive','ForegroundPolarity','dark','Sensitivity',0.5)
;
SE = fspecial('disk',1);
gssmooth = imfilter(gsf,SE,"replicate");
BWfiltrada =
imbinarize(gssmooth,'adaptive','ForegroundPolarity','dark','Sensitivity'
,0.5);
figure(3)
subplot(5,5,k)
imshowpair(gsf,BWf,'montage')
Suma1 = sum(BWf,2);
Suma2 = sum(BWfiltrada);
```

```
Ruido = (Suma1 - Suma2');
figure(4)
subplot(5,5,k)
plot(Suma1)
hold on
plot(Suma2)
xfiltrada = imfilter(f,SE,"replicate");
figure(5)
subplot(5,5,k)
imshowpair(f,xfiltrada,'montage')
ArtefactosImagen = imbinarize((BWf -
BWfiltrada),'adaptive','ForegroundPolarity','dark','Sensitivity',0.5);
figure(6)
subplot(5,5,k)
imshowpair(f,ArtefactosImagen,'montage')
end
```

Bibliografía

[1] Finagro, «El Momento del Agro,» 30 Octubre 2020. [En línea]. Available: https://www.finagro.com.co/noticias/el-momento-del-agro.

[2] Ministerio de Agricultura y Desarrollo Rural, «minagricultura,» Marzo 2019. [En línea]. Available: https://sioc.minagricultura.gov.co/Papa/Documentos/2019-03-31%20Cifras%20Sectoriales.pdf.

[3] Ministerio de Agricultura y Desarrollo Rural, «Minagricultura,» Junio 2020. [En línea]. Available: https://sioc.minagricultura.gov.co/Papa/Documentos/2020-06-30%20Cifras%20Sectoriales.pdf.

[4] C. d. C. d. Bogotá, V. d. F. Empresarial y P. d. A. A. y. Agroindustrial, «Centro de Información Empresarial (CIEB),» 2015. [En línea]. Available: http://hdl.handle.net/11520/14306.

[5] H. Torres, «Cipotato.org,» Mayo 2002. [En línea]. Available: http://cipotato.org/wp-content/uploads/2002/05/002485-1.pdf.

[6] W. Pérez y G. Forbes, «fao.org,» julio 2011. [En línea]. Available: http://www.fao.org/3/as407s/as407s.pdf.

[7] T. -Y. Lee, J. -Y. Yu, Y. -C. Chang y J. -M. Yang, «Health Detection for Potato Leaf with Convolutional Neural Network,» de *2020 Indo – Taiwan 2nd International Conference on Computing, Analytics and Networks (Indo-Taiwan ICAN)*, 2020.

[8] F. Liu y Z. Xiao, «Disease Spots Identification of Potato Leaves in Hyperspectral Based on Locally Adaptive 1D-CNN,» de *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 2020.

[9] F. Islam, M. N. Hoq y C. M. Rahman, «Application of Transfer Learning to Detect Potato Disease from Leaf Image,» de *2019 IEEE International Conference on*

Robotics, Automation, Artificial-intelligence and Internet-of-Things (RAAICON), 2019.

[10] P. Sharma, P. Hans y S. C. Gupta, «Classification Of Plant Leaf Diseases Using Machine Learning And Image Preprocessing Techniques,» de *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 2020.

[11] A. Anton, S. Rustad, G. Shidik y A. Syukur, «Classification of Tomato Plant Diseases Through Leaf Using Gray-Level Co-occurrence Matrix and Color Moment with Convolutional Neural Network Methods,» de *Smart Trends in Computing and Communications: Proceedings of SmartCom 2020.*, Singapore, Springer, 2020.

[12] N. Ruedeeniraman, M. Ikeda y L. Barolli, «Performance Evaluation of VegeCare Tool for Potato Disease Classification,» de *Advances in Networked-Based Information Systems*, vol. 1264, Cham, Springer, 2021, pp. 470-478.

[13] M. Agarwal, A. Sinha, S. Gupta, D. Mishra y R. Mishra, «Potato Crop Disease Classification Using Convolutional Neural Network,» de *Smart Systems and IoT: Innovations in Computing*, vol. 141, Singapore, Springer, 2020, pp. 391-400.

[14] G. Madhulatha y O. Ramadevi, «Recognition of Plant Diseases using Convolutional Neural Network,» de *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India, 2020.

[15] H. Pant, J. Pant, P. K. Pant, D. Singh y M. C. Lohani, «Web based gui detector to recognise tomato plant leaf disease,» *International Journal of Emerging Trends in Engineering Research,* pp. 5769-5775, 2020.

[16] K. P. Ferentinos, «Deep learning models for plant disease detection and diagnosis,» *Computers and Electronics in Agriculture,* vol. 145, pp. 311-318, 2018.

[17] M. Islam, A. Dinh, K. Wahid y P. Bhowmik, «Detection of potato diseases using image segmentation and multiclass support vector machine,» de *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2017.

[18] Aparajita, R. Sharma, A. Singh, M. K. Dutta, K. Riha y P. Kriz, «Image processing based automated identification of late blight disease from leaf images of potato crops,» de *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*, 2017.

[19] M. A. Jasim y J. M. AL-Tuwaijari, «Plant Leaf Diseases Detection and Classification Using Image Processing and Deep Learning Techniques,» de *2020 International Conference on Computer Science and Software Engineering (CSASE)*, 2020.

[20] M. Al-Amin, T. A. Bushra y M. N. Hoq, «Prediction of Potato Disease from Leaves using Deep Convolution Neural Network towards a Digital Agricultural System,» de *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, 2019.

[21] FAO, «Año Internacional de la Papa,» 2008. [En línea]. Available: http://www.fao.org/potato-2008/es/lapapa/index.html.

[22] C. C. Westcott, «Chapter 1 - Principles of pH Measurements,» de *PH Measurements*, Elsevier, 1978, pp. 1-16.

[23] A. Fuentes, S. Yoon, S. Kim y D. Park, «A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition,» *Sensors (Basel, Switzerland),* vol. 17, 2017.

[24] N. Y. Grisales-Vásquez y J. M. Cotes-Torres, «General and Specific Combinatorial Aptitude in a F1 Population of Solanum phureja with Resistance to Phytophthora infestans,» *American Journal of Potato Research,* vol. 96, pp. 55-61, 2019.

[25] R. A. Krishnaswamy, M. Prabhakar, R. Purushothaman y S. Cezary, «Chapter nine - Crop disease classification using deep learning approach: an overview and a case study,» de *Deep Learning for Data Analytics*, Academic press, 2020, p. 173.195.

[26] S. Huang, W. Liu, F. Qi y K. Yang, «Development and Validation of a Deep Learning Algorithm for the Recognition of Plant Disease,» de *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019.

[27] N. Buduma y N. Locascio, Fundamentals of deep learning: Designing next-generation machine intelligence algorithms, O'Reilly Media, Inc., 2017.

[28] S. Haykin, Neural Networks and Learning Machines, Pearson, 2009.

[29] W. Steven y C. Narciso, «Artificial Neural Networks,» de *Encyclopedia of Physical Science and Technology*, Third Edition ed., New York, Academic Press, 2003, pp. 631-645.

[30] A. Abraham, «Artificial neural networks,» *Handbook of measuring system design,* pp. 901-908, 2005.

[31] R. Venkatesan y B. Li, Convolutional Neural Networks in Visual Computing A Consice Guide, Boca Raton: CRC Press, 2018.

[32] H. H. Aghdam y E. J. Heravi, Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification, Springer, 2017.

[33] J. A. López Sotelo, Deep Learning Teoria y Aplicaciones, Bogotá: AlfaOmega Colombiana S.A - Alpha Editorial, 2021.

[34] A. Kumar y M. Vani, «Image Based Tomato Leaf Disease Detection,» de *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2019.

[35] M. T. T., S. M. H., S. R. A., R. M. S. y M. S. Islam, «Leaves Diseases Detection of Tomato Using Image Processing,» de *2019 8th International Conference System Modeling and Advancement in Research Trends (SMART)*, 2019.

[36] M. Munnangi, «Crops: Plant Disease Identification Using Mobile App,» 18 Octubre 2019. [En línea]. Available: https://towardsdatascience.com/crop-plant-disease-identification-using-mobile-app-aef821d1a9bc.

[37] N. Buduma y L. Nicholas, Fundamentals of Deep Learning, Primera ed., Beijing, Boston, Farnham, Sebastopol, Tokyo: O'Reilly, 2017.

[38] R. Yamashita, M. Nishio, R. Do y K. Togashi, «Convolutional neural networks: an overview and application in radiology,» *Insights into Imaging,* vol. IX, pp. 611-629, 2018.

[39] R. C. Gonzalez y R. E. Woods, Digital Image Processing, Pearson Prentice Hall, 2008.

[40] D. Hughes y M. Salathe, «An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing,» pp. 1-13, 2015.

[41] S. Dasgupta, S. Rakshit, D. Mondal y D. Kole, «Detection of Diseases in Potato Leaves Using Transfer Learning,» de *Computational Intelligence in Pattern Recognition. Advances in Intelligent Systems and Computing*, vol. 999, Singapoore, Springer, 2020, pp. 675-684.

[42] D. Tiwari, M. Ashish, N. Gangwar, A. Sharma, S. Patel y S. Bhardwaj, «Potato Leaf Diseases Detection Using Deep Learning,» de *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Mandurai, India, 2020.

[43] Y. Toda y F. Okura, «How Convolutional Neural Networks Diagnose Plant Disease,» *Plant Phenomics,* pp. 1-14, 2019.

[44] International Telecommunication Union, *Recommendation ITU-R BT.601-7,* Geneva: ITU, 2011.

[45] C. G. Kanan C, «Color-to-Grayscale: Does the Method Matter in Image Recognition?,» *PLoS ONE 7(1),* nº e29740, 2012.

[46] S. Roy, A. Mitra y S. K. Setua, «Color & grayscale image representation using multivector,» de *Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*, 2015.

[47] D. P. Kingma y J. Ba, «Adam: A method for stochastic optimization,» *arXiv preprint arXiv,* vol. 1412, nº 6980, 2014.

[48] OIML, «Organisation Internationale de Mètrologie Légale,» 2008. [En línea]. Available: https://www.oiml.org/en/files/pdf_g/g001-100-e08.pdf.