Universidad Nacional de San Antonio Abad del Cusco

Facultad de Ingeniería Eléctrica, Electrónica, Informática y Mecánica

Escuela Profesional de Ingeniería Informática y de Sistemas

"EVALUACIÓN DE ARQUITECTURAS DEEP LEARNING PARA IDENTIFICAR ANOMALÍAS POR DEFICIENCIA DE MACRONUTRIENTES EN PERSEA AMERICANA VARIEDAD HASS."

Tesis presentada por:

Br. Gregori Vicente Huamán Cruz
Br. Lucero Betzabe Machaca Quispe

Para optar al Titulo Profesional de:

Ingeniero Informático y de Sistemas

Asesor:

Mg. Julio Cesar Carbajal Luna

Cusco-Perú
2019

Dedicatoria

A nuestros queridos padres que con amor, confianza y apoyo incondicional permitieron que hoy logremos alcanzar una de nuestras metas.

Br. Lucero Betzabe Machaca Quispe
Br. Gregori Vicente Huamán Cruz

Agradecimientos

A Dios, por darnos la vida, salud y guiar nuestro camino. Ser el apoyo y fortaleza en aquellos momentos de dificultad y de debilidad.

A nuestro asesor, por darnos su confianza y brindarnos su apoyo incondicional en todo este proceso.

A nuestros padres, por enseñarnos que el trabajo duro tiene sus recompensas, por los valores y principios que nos han inculcado. A ellos a quienes les debemos nuestras vidas.

A nuestros hermanos(as), por estar siempre presentes, su cariño y apoyo incondicional.

A nuestros amigos, por acompañarnos en buenos y malos momentos.

Br. Lucero Betzabe Machaca Quispe
Br. Gregori Vicente Huamán Cruz

Resumen

La fertilidad del suelo es vital para el desarrollo de los cultivos de Persea Americana y una beneficiosa producción de frutos, este desarrollo se ve afectado por la deficiencia de nutrientes (macro y micro nutrientes) en el suelo. Las características tempranas se presentan en las hojas provocando decoloraciones, manchas y en algunos casos necrosis. En los macronutrientes se consideran seis clases críticas para el cultivo como son Calcio, Potasio, Magnesio, Nitrógeno, Fósforo y Azufre, la dificultad de identificar al macronutriente que ocasiona la aparición de anomalías en las hojas corresponde a un problema de clasificación de imágenes. Las redes neuronales convolucionales han demostrado un gran desempeño en las tareas de visión por computadora, esencialmente en la clasificación de imágenes por este motivo la solución propone emplear las arquitecturas de redes neuronales profundas Inception-v3, DenseNet169 y MobileNet por su alto rendimiento. Para entrenar las CNNs se construyó un dataset original y primero en su tipo de 6,090 imágenes distribuidas en siete categorías que se recolectaron en cuatro periodos, en el sector de Molinopata, Abancay - Apurímac, aún con esta cantidad de imágenes es insuficiente para entrenar una CNN desde cero. Por esta razón se empleó la transferencia de aprendizaje en la solución como estrategia de entrenamiento, adicionalmente beneficia el desarrollo de la solución reduciendo el costo computacional y los tiempos de entrenamiento. Se construyó un prototipo basado en un arquitectura API y desplegado en el servidor local con finalidad de mostrar los resultados de manera dinámica para un usuario final.

Los resultados obtenidos de las métricas de evaluación aplicadas al subconjunto de validación de *Persea Americana Hass Dataset* proporcionan un análisis del comportamiento de las CNNs en la tarea clasificación de las hojas de Persea Americana variedad Hass. Se obtuvo tasas de acierto del 85.86 %, 84.64 % y 83.29 % de Inception-v3, MobileNet y DenseNet169 respectivamente. Resultando ser la CNN Inception-v3 la sobresaliente para el problema de investigación.

Palabras clave: *Persea Americana, Hass, Macronutrientes, Redes Neuronales Convolucionales, CNN, Transferencia de aprendizaje, Transfer Learning, Dataset, Inception-v3, DenseNet169, MobileNet, API.*

Abstract

Ground fertility is essential for the growth of Persea Americana crops and beneficial fruit production, this development is affected by nutrient deficiency (macro and micronutrients) on the ground. Early characteristics are presented in the leaves causing discoloration, stains and in some cases necrosis. Six macronutrients are considered critical for growth are calcium, potassium, magnesium, nitrogen, phosphorus and sulfur, the difficulty of identifying the macronutrient causes the appearance of anomalies corresponding to a problem of image classification leaves.Convolutional neural networks have shown great performance on tasks of computer vision, essentially image classification for this reason, the solution proposed use deep architectures Inception-v3, DenseNet169 and MobileNet for high performance. To train CNN an original data set of 6,090 images was built, distributed in seven categories that were collected in four periods in the sector of Molinopata, Abancay- Apurimac, even with this amount of images is not enough to train a CNN from scratch. For this reason the transfer of learning was used in solution as training strategy further benefits the development of the solution by reducing the computational cost and training time. A prototype was developed based on the architecture of the API and deployed on a local server to show the results in a dynamic interface for final user.

The results of the evaluation metrics applied to the validation subset Persea Americana Hass Dataset provide an analysis of the performand of CNNs in the classification of persea americana leaves. Success rates of 85.86 %, 84.64 % and 83.29 % of Inception-v3, MobileNet and DenseNet169 respectively were obtained. Inception-v3 is the architecture that stands out.

Keywords: *Persea Americana, Hass, Macronutrients, Convolutional Neural Networks, CNN, Transfer Learning, Dataset, Inception-v3, DenseNet169, MobileNet, API.*

Índice general

Índice de figuras

Índice de tablas

Nomenclatura

Acrónimos

AJAX Asynchronous JavaScript y XML

ANN Artifical Neural Network

API Application Programming Interface

BN Normalizacion de Lotes

BSON Binary JSON

CMYK Cyan, Magenta, Yellow, Key

CNN Convolutional Neural Networks

COMEX Sociedad de Comercio Exterior del Perú

CPU Central Processing Unit

CRUD Create, Read, Update and Delete

CSS Cascading Style Sheets

CVPR Best Paper Awards

DDR Double Data Rate

DOM Document Object Model

eFV Exponential Family Fisher Vectors

FLOPs Floating Point Operations per Second

Gb Gigabyte

GPU Graphics Processing Unit

GUI Graphical User Interface

HSV Hue, Saturation, Value

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IA Inteligencia Artificial

ILSVRC ImageNet Large Scale Visual Recognition Challenge

ISBI International Symposium on Biomedical Imaging

JSON JavaScript Object Notation

MINAGRI Ministerio de Agricultura y Riego

ORM Object-Relational mapping

RELU Rectified Linear Unit

RGB Red - Green - Blue

RNN Recurrent Neural Networks

URL Localizador Uniforme de Recursos

Capítulo 1

Aspectos Generales

1.1. Planteamiento del Problema de Investigación

1.1.1. Descripción del problema de investigación.

La Persea Americana comúnmente conocida como palta o aguacate, es fruto del árbol originario de Centroamérica, pertenece a la familia de las lauráceas. Su cultivo se diversificó por todo el continente americano, dando origen a una gran variedad; una de las principales es la variedad Hass, considerada la variedad con un alto nivel de exportación debido a su excelente calidad de pulpa y cáscara gruesa que permite tolerar bien el transporte a largas distancias.

La Sociedad de Comercio Exterior del Perú (COMEX Perú), destaca la palta como el "oro verde" de las exportaciones peruanas, al lograr un crecimiento sostenido entre enero y diciembre del 2017, posicionando al Perú como el segundo mayor exportador de palta fresca, después de México y por encima de Chile.

La fertilidad del suelo es vital para el desarrollo del árbol y una provechosa producción de frutos; este desarrollo es afectado por deficiencias nutricionales (macro y micronutrientes) que originan problemas como manchas y decoloraciones en hojas, deformacione en el fruto, entre otros; Por lo tanto, de no adoptar medidas necesarias causan daños irreparables.

Las características tempranas de la deficiencia de un macronutriente se manifiestan en las hojas de la Persea Americana variedad Hass e identificar al macronutriente causante, representa una dificultad para el agricultor. La recolección y agrupación de hojas según patrones similares generan información trascendente para investigaciones.

La identificación de las deficiencias de macronutrientes en la Persea Americana variedad Hass, corresponde a un problema de clasificación de imágenes en el área de visión por

computadora. La aplicación de técnicas de *Deep Learning* empleando arquitecturas de redes neuronales convolucionales, demostraron un eficiente desempeño en tareas similares. Es fundamental determinar la arquitectura convolucional que provea un buen desempeño para identificar las deficiencias de macronutrientes en la Persea Americana varidad Hass.

La aplicación de inteligencia artificial en el sector agrícola del Perú aún es limitada, adicionado a la escasa existencia de bancos de datos que faciliten a los investigadores realizar trabajos al respecto; por esta razón el presente proyecto de investigación propone la elaboración de un conjunto de datos recolectados de hojas de Persea Americana variedad Hass y la implementación de un clasificador de imágenes mediante Deep Learning, para identificar anomalías por deficiencia de macronutrientes en cultivos de Persea Americana variedad Hass.

1.1.2. Formulación del problema de investigación.

¿Cuál es la arquitectura de Deep Learning con mejor desempeño para la identificación de deficiencia de macronutrientes en la Persea Americana variedad Hass?

1.2. Objetivos de la Investigación

1.2.1. Objetivo general.

Determinar la arquitectura Deep Learning con mejor desempeño para identificar las deficiencias de macronutriente en la Persea Americana variedad Hass.

1.2.2. Objetivos específicos.

- Identificar las características que se presentan en las hojas por la deficiencia de macronutrientes en la Persea Americana variedad Hass.

- Elaborar un conjunto de datos (Dataset) ideal para la identificación de anomalías generadas por deficiencia de macronutrientes en la Persea Americana variedad Hass.

- Generar nuevos modelos de redes neuronales convolucionales adoptando la transferencia de aprendizaje para identificar las anomalías por deficiencia de macronutrientes en la Persea Americana variedad Hass.

- Comparar los resultados obtenidos de las arquitecturas de redes neuronales convolucionales para identificar anomalías generadas por deficiencia de macronutrientes en la Persea Americana variedad Hass.

- Extrapolar los resultados en un conjunto de datos diferente aplicando los modelos de redes neuronales convolucionales entrenados.

- Desarrollar un prototipo de software para la identificación de anomalías generadas por deficiencia de macronutrientes en la Persea Americana variedad Hass.

1.3. Justificación de la Investigación

En los últimos años el Perú se ha convertido en uno de los principales exportadores de Persea Americana, este crecimiento a impulsado su cultivo en diversas regiones del país. La fertilidad del suelo es fundamental para una favorable producción de Persea Americana; las características tempranas de la deficiencia de un macronutriente se manifiestan en las hojas y clasificarlas posee un grado de dificultad. La recolección de imágenes de hojas que exhiban estas características y la implementación de arquitecturas de redes neuronales convolucionales profundas, constituyen la solución propuesta por el trabajo de investigación. La aplicación de redes neuronales convolucionales demostraron un eficiente rendimiento en tareas de visión por computadora, convirtiéndose en la técnica más empleada para resolver problemas de clasificación de imágenes. Facilitar el proceso de identificación del macronutriente que se manifiesta, beneficia al agricultor para que pueda adoptar medidas sobre la fertilidad del suelo en sus cultivos. Por otro lado los resultados obtenidos pueden ser de utilidad para trabajos de investigación futuros.

1.4. Alcances y Limitaciones

Alcances

- Identificar el macronutriente aplicando redes convolucionales profundas.

- Las anomalías originadas por deficiencia de macronutrientes corresponden a las descritas en el Manual Técnico de Buenas Prácticas en el Cultivo de Palto, MINAGRI - 2010.

- La construcción del dataset se divide en siete clases considerando hojas asintomáticas y seis deficiencias por macronutrientes visibles en las hojas.

Limitaciones

- El prototipo está implementado en una arquitectura API.
- El proyecto de investigación no considera otras variedades de Persea Americana; solo considera la variedad Hass.
- La recolección de imágenes se realizó en el sector: Molinopata, provincia: Abancay, región: Apurímac; en cuatro etapas.
- La clasificación de imágenes se realizó aplicando estrategias y arquitecturas de redes convolucionales.

Capítulo 2

Marco Teórico

2.1. Antecedentes de la Investigación

2.1.1. Antecedentes internacionales.

[Lopez Pacheco and Liu, 2017](Tesis de Maestría - Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional - México). En su estudio "Identificación de sistemas no lineales con redes neuronales convolucionales" describe una estructura de una red neuronal convolucional (CNN) para la identificación de sistemas no lineales, por medio de dos paradigmas de aprendizaje, supervisado y no supervisado. Este estudio tuvo tres fases. La primera fase trata sobre la arquitectura de la red neuronal que propone el autor y las ecuaciones matemáticas del modelo no supervisado, el uso de mínimos cuadrados para entrenar la red y sus correspondientes simulaciones, comparando el mejor desempeño de errores cuando hay ruido y no, en datos de entrenamiento, existiendo muy poca diferencia. En la segunda fase para el aprendizaje supervisado, se mencionan métodos de aprendizaje en línea, se realizaron modificaciones del algoritmo de *backpropagation* para entrenar la red, al igual que en la fase anterior se presentó dichas simulaciones realizadas con esta metodología de aprendizaje para los mismos sistemas de prueba pero agregando más parámetros a la red y realizando comparaciones entre ellas. Finalmente fue demostrando las ventajas existentes de las redes neuronales convolucionales para la identificación de sistemas no lineales cuando hay ruido en los datos.

[Galárraga Cañizares, 2017] (Tesis de Maestría - Universidad Politécnica de Madrid - España). En su estudio "Clasificador de hojas mediante Deep Learning" describe la generación de un clasificador de enfermedades en las plantas empleando imágenes, que utiliza un método basado en aprendizaje profundo como algoritmo de clasificación. El proyecto utilizó la base de datos pública Plant Village, que dispone de un total de 10.263 imágenes entre hojas enfermas y sanas. Este estudio tuvo tres fases elementales. La primera fase fue la segmentación de las imágenes cuyo objetivo es aislar las regiones de interés en la imagen. La segunda fase la búsqueda de los mejores ajustes de los parámetros de configuración del algoritmo de aprendizaje profundo para el entrenamiento de una red neuronal convolucional con dos especies de cultivos (tomates y uvas) y 13 enfermedades, obteniendo un modelo entrenado con una exactitud del 98,37 % para el diagnóstico de las plantas enfermas y sanas, se obtuvo un clasificador capaz de clasificar correctamente 2,041 imágenes de un total de 2,053 ejemplos preclasificados por un experto. Y finalmente la distribución del modelo en una herramienta de simulación elaborada en Matlab con interfaz gráfica.

[Redolfi et al., 2016](Paper - VIII Congreso Argentino de AgroInformática (CAI) Argentina.) En su estudio "Clasificación de variedades de semillas de trigo usando visión por computadora" abordo el problema de identificación de variedades de semillas de trigo y propusieron el uso de técnicas actuales de clasificación de imágenes como son: Vectores de Fisher de la Familia Exponencial (eFV) y Redes Neuronales Convolucionales (CNN). Se elaboró un dataset compuesto por 315 muestras de seis variedades diferentes de trigo recolectado. El uso de CNN logró mejores resultados, alcanzó una exactitud del 95 %, demostrando que la disponibilidad de un método de clasificación automático de semillas acelera los procesos de evaluación y permite que sean realizados en diferentes etapas del proceso de producción de manera simple y con bajo costo.

2.1.2. Antecedentes nacionales.

[Coronado Pérez, 2018](Tesis de Maestría - Universidad Nacional de San Agustín - Arequipa). En su estudio titulado "Reconocimiento de patrones en imágenes no dermatoscopicas para la detección de enfermedades malignas en la piel, utilizando redes neuronales convolutivas y autocodificadores", propone un método para el reconocimiento de lesiones en la piel y lograr identificar lesiones malignas en imágenes no dermatoscópicas. Este estudio fue dividido en tres fases. La primera fase tuvo como objetivo construir un repositorio de imágenes con casos de enfermedades de piel, categorizadas como benignas, pre-malignas,

a partir de datos públicos. La segunda fase tuvo como objetivo el desarrollo de un método para la extracción de características basadas en redes neuronales convolutivas y un método de clasificación de imágenes basado en auto codificadores. Y finalmente fue evaluado el método de clasificación, obteniendo un modelo entrenado con una precisión de 91.56 % en comparaciones con resultados obtenidos en el estado del arte presentado en este estudio, Se logró superar al primer lugar del concurso ISBI-2016, de acuerdo con el indicador de sensibilidad, la arquitectura de red convolucional propuesta fue VGG19 que identifica mejor los casos de cáncer de piel.

[Coronel Zegarra and Calderón Niquín, 2016](Paper - VIII Congreso Internacional de Computación y Telecomunicaciones (COMTEL) - Lima). En su estudio "Sistema Online Basado en Verificación Facial desde Dispositivos Móviles empleando Redes Neuronales Convolucionales" realizó un proceso de verificación facial, fundamentado en seguir el proceso de detectar rostros, alinearlos, representarlos y clasificarlos. Se utilizó las redes neuronales convolucionales como método para la extracción de vectores de características, los vectores son representativos de cada rostro y permiten realizar una comparación que devuelva una distancia entre ambos. Asimismo, para el desarrollo se utilizó OpenSource de OpenFace que incluye un modelo preentrenado de red neuronal convolucional basado en FaceNet. Se realizaron pruebas de efectividad del sistema con una muestra de 484 comparaciones de 22 personas. El porcentaje total de resultados correctos o exactitud de todos los casos fue de 86 % siendo 417 resultados acertados de un total de 484.

2.2. Marco Conceptual

2.2.1. Procesamiento digital de imágenes.

El procesamiento digital de imágenes está presente en aplicaciones industriales, ciencias médicas, biometría e identificación, agricultura, ganadería, satélites de observación terrestre, como en tantas otras tareas de distinta índole en la sociedad. Es por ello que hoy en día se enfatiza mucho en desarrollar nuevas y mejores técnicas para llevar a cabo estos tratamientos en las diversas actividades donde se las requiera.[Gonzalez and Woods, 2002] Es una tecnología asociada a las ciencias de la computación y, por tanto, cabe en ella como una proyección del término Visión Artificial dentro del ámbito de la Inteligencia Artificial.[Jaime and Enrique, 2002]

2.2.1.1. Imagen digital.

Una imagen puede considerarse como el conjunto de puntos de colores, es decir, una sucesión coherente de puntos que conforman una matriz de información para el uso digital. Estos puntos se denominan 'píxeles' *(picture element)*. Un píxel es la menor unidad homogénea en color que forma parte de una imagen digital [Gonzalez and Woods, 2002].

La imagen digital es una función bidimensional $f(x,y)$, cuyo valor es el grado de iluminación o intensidad de luz en el espacio de coordenadas (x,y) de la imagen para cada punto. El valor de la función depende de la cantidad de iluminación $i(x,y)$ y reflexión $r(x,y)$. El producto de las dos funciones proporciona la función $f(x,y)$. La iluminación depende de la fuente de luz, mientras que la reflexión de las características del objeto en la escena [Escalera Hueso, 2001].

2.2.1.2. Colores.

El utilizar color en visión es importante ya que puede ayudar a la extracción de características e identificación de objetos en la imagen. Existen diferentes sistemas para la representación de colores, los más conocidos son:

- ***RGB(Red - Green - Blue).*** Es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios. El modelo de color RGB no define por sí mismo lo que significa exactamente rojo, verde o azul, por lo que los mismos valores RGB pueden mostrar colores notablemente diferentes en distintos dispositivos que usen este modelo de

color. Aunque utilicen un mismo modelo de color, sus espacios de color pueden variar considerablemente. [Baker and Hearn, 1995]. Figura 4.5(b)

- **HSV(*Hue, Saturation, Value).*** El modelo HSV fue creado en 1978 por Alvy Ray Smith. Se trata de una transformación no lineal del espacio de color RGB, y se puede usar en progresiones de color. [Baker and Hearn, 1995]. Figura 2.1(b)

- ***CMYK(Cyan, Magenta, Yellow, Key).*** Es un modelo de color sustractivo que se utiliza en la impresión en colores. Es la versión moderna y más precisa del antiguo modelo tradicional de coloración (RYB), que se utiliza todavía en pintura y artes plásticas. Permite representar una gama de colores más amplia que este último, y tiene una mejor adaptación a los medios industriales. [Baker and Hearn, 1995]. Figura 2.1(c)

(a) RGB (b) HSV (c) CMYK

Figura 2.1 Sistema de colores
Fuente: https://bit.ly/2DTlB8K, https://bit.ly/2AAl20G, https://bit.ly/2r9uLGV

2.2.1.3. Pasos del procesamiento digital de imágenes.

Los métodos de procesamiento digital de imágenes se pueden distinguir dos amplias categorías: Métodos cuyos *inputs y outputs* son imágenes y Métodos cuyos *inputs* pueden ser imágenes, pero cuyos *outputs* son atributos extraídos de esas imágenes. Esta organización se resume en la Figura 2.2. El diagrama no implica que todos los procesos se apliquen a una imagen. Más bien, la intención es transmitir una idea de todas las metodologías que se pueden aplicar a las imágenes para diferentes propósitos y posiblemente con diferentes objetivos. [Gonzalez and Woods, 2002].

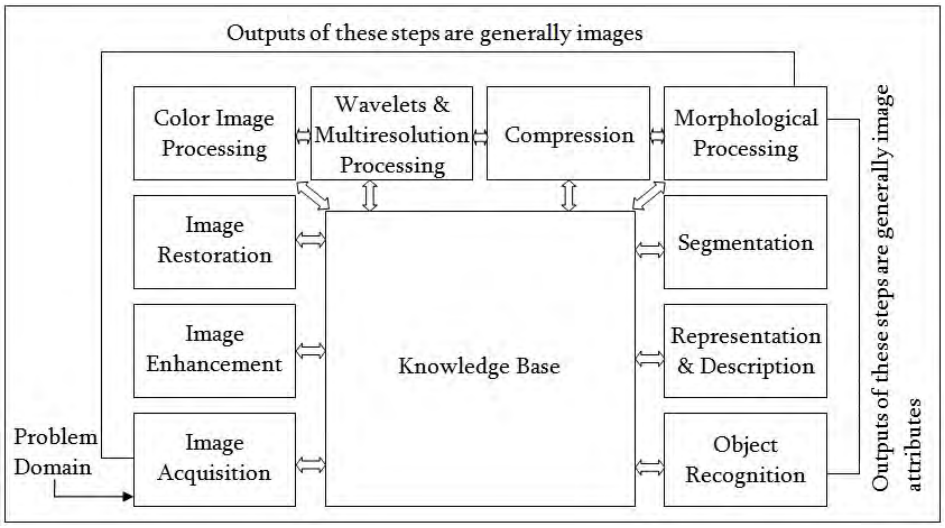


Figura 2.2 Pasos fundamentales en el procesamiento digital de imágenes.
Fuente: [Gonzalez and Woods, 2002]

Adquisición de imágenes *(Image Acquisition)* Este es el primer paso, la adquisición de imágenes podría recibir una imagen que ya está en formato digital, o requiere de un sensor de imágenes, cuyas señales producidas deben ser digitalizadas. Dentro de esta etapa existen múltiples factores que deben considerarse en el diseño de una captura adecuada como la cámara, la óptica, la tarjeta de adquisición, el ordenador, el software de adquisición, entre otros. Además hay que tener en cuenta factores relacionados con la escena como la iluminación, el fondo, la posición de la cámara, entre otros [Gonzalez and Woods, 2002].

Realce de la imagen *(Image Enhancement)* en esta etapa se usan técnicas de mejora para resaltar detalles que están ocultos, o simplemente resaltar ciertas características de interés en una imagen. Tales como: cambio de brillo y contraste [Gonzalez and Woods, 2002].

Restauración de la imagen *(Image Restoration)* este paso también se relaciona con la mejora en la apariencia de una imagen. Sin embargo, a diferencia de la mejora, que es subjetiva, la restauración de la imagen es objetiva, en el sentido de que las técnicas de restauración tienden a basarse en modelos matemáticos o probabilísticos de degradación de la imagen. [Gonzalez and Woods, 2002].

Procesamiento de imágenes en color *(Color Image Processing)* este paso tiene un uso muy frecuente, la imagen en color puede dar información adicional a la imagen en tonos de gris, dependiendo de la aplicación. Para procesar una imagen en color resulta imprescindible manejar los siguientes conceptos:

- Fundamentos del color.
- Modelos básicos del color.
- Pseudocolor.

[Gonzalez and Woods, 2002]

Ondeletas y procesamiento multi-resolución *(Wavelets and Multi-Resolution Processing)* Las wavelets son la base para representar imágenes en varios grados de resolución. Las imágenes se subdividen sucesivamente en regiones más pequeñas para la compresión de datos y para la representación piramidal.

Comprensión de imágenes *(Compression)* se trata precisamente, de comprimir la imagen con dos propósitos fundamentales:

- Reducir su tamaño para su almacenamiento.
- Reducir su ancho de banda para su transmisión.

[Gonzalez and Woods, 2002]

Procesos morfológicos *(Morphological Processing)* son el conjunto de herramientas usadas para extraer aquellos componentes de la imagen que son útiles para la representación y descripción de las formas [Gonzalez and Woods, 2002].

Segmentación *(Segmentation)* es un proceso que se utiliza para extraer o aislar objetos del resto de la imagen para su posterior análisis. Se da en dos sentidos:

- Segmentación autónoma, para facilitar procesos subsecuentes.
- Salidas, pixeles borde o que indican la frontera entre objetos, esquinas, texturas, entre otros.

[Gonzalez and Woods, 2002]

Representación y descripción *(Representation and description)* la selección de características (descripción), el proceso consiste en extraerlas en:

- Representación como bordes: características de forma externa como esquinas, inflexiones, entre otros.
- Representaciones como regiones: características de propiedades internas como texturas, formas de esqueleto, entre otros.

[Gonzalez and Woods, 2002]

Reconocimiento *(Recognition)* es posible asignar una etiqueta a un objeto, basados en la información provista por los descriptores, o bien asignar un significado a un grupo de objetos ya reconocidos [Gonzalez and Woods, 2002].

Conocimiento acerca del dominio del problema *(Knowledge Base)* está codificado dentro del sistema de procesamiento de imágenes en forma de una base de datos de conocimiento. Este puede ser tan simple como detallar las regiones de donde la información de interés esta situada, o bien, compleja definiendo una lista de características tomadas de diferentes contextos. [Gonzalez and Woods, 2002].

En términos generales mientras la complejidad de una tarea de procesamiento digital de imagen crece, el número de procesos requeridos para resolver el problema también crece. [Gonzalez and Woods, 2002]

2.2.2. Machine Learning.

Machine Learning, también conocido como aprendizaje automático es una rama de la inteligencia artificial (ver Figura 2.3), cuyo objetivo es el desarrollo de técnicas que proporcionan a los ordenadores la capacidad de aprender, sin ser explícitamente programados [Samuel, 1967]. Más tarde [Mitchell, 1997], define el aprendizaje automático: “Dice que un programa de computación aprende de la experiencia E con respecto a una tarea T y alguna medida de rendimiento P, si es que el rendimiento en T, medido por P, mejora con la experiencia E”.

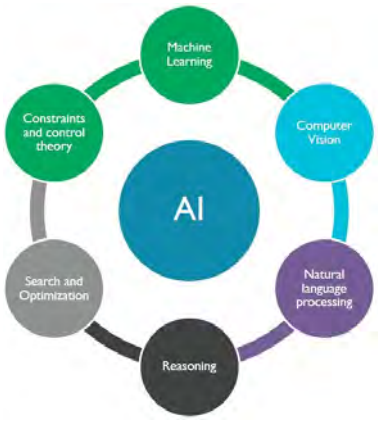


Figura 2.3 Ramas de la inteligencia artificial
Fuente: https://bit.ly/2PhWPBn

El aprendizaje automático es una tecnología que permite a las computadoras aprender directamente de ejemplos y experiencias en forma de datos. Los enfoques tradicionales de programación se basan en reglas codificadas, que establecen cómo resolver un problema, paso a paso. En contraste, los sistemas de aprendizaje automático se configuran como una tarea y se les da una gran cantidad de datos para usar como ejemplos de cómo se puede lograr esta tarea o para detectar patrones. Luego, el sistema aprende la mejor manera de lograr la salida deseada [Royal Society (Great Britain), 2017].

Objetivos de Machine Learning.

El campo de machine learning se organiza en torno a tres focos principales de investigación:

- **Estudio orientado a tareas:** el desarrollo y análisis de sistemas de aprendizaje para mejorar el rendimiento en un conjunto predeterminado de tareas (también conocido como " enfoque de ingeniería")

- **Simulación cognitiva:** la investigación y simulación por computadora de los procesos de aprendizaje humano.

- **Análisis teórico:** la exploración teórica del espacio de posibles métodos de aprendizaje y algoritmos independientes de la aplicación.

Aunque muchos esfuerzos de investigación se impulsan principalmente hacia uno de estos objetivos, el progreso hacia uno a menudo conduce al progreso hacia otro. Por ejemplo, para investigar el espacio de los posibles métodos de aprendizaje, un punto de partida razonable puede ser considerar el único ejemplo conocido de comportamiento de aprendizaje sólido, los humanos (y quizás otros sistemas biológicos). De manera similar, las investigaciones psicológicas del aprendizaje humano pueden ser ayudadas por un análisis teórico que puede sugerir varios modelos de aprendizaje plausibles. La necesidad de adquirir una forma particular de conocimiento en un estudio orientado a tareas puede generar un nuevo análisis teórico o plantear la pregunta: "¿Cómo adquieren los humanos esta habilidad específica (o conocimiento)?" esta tricotomía de objetivos mutuamente desafiantes y de apoyo es un reflejo de todo el campo de la inteligencia artificial, donde la búsqueda de sistemas expertos, estudios cognitivos y teóricos proporcionan fertilización cruzada de problemas e ideas [Michalski et al., 2013].

2.2.2.1. Tipos de Machine Learning.

El aprendizaje automático se divide generalmente en dos tipos principales:

- En el enfoque del **aprendizaje predictivo o supervisado**, el objetivo es aprender un mapeo de los *inputs* x a los *outputs* y, dado un conjunto etiquetado de pares de salidas de entrada $D = \{(x_i, y_i)\}_{i=1}^{N}$, donde D se denomina **conjunto de entrenamiento**, y N es el número de ejemplos de entrenamiento. En la configuración más simple, cada *input* de entrenamiento x_i es un vector de números D-dimensionales, que representa, digamos, la altura y el peso de una persona. Estos se llaman **características, atributos o co-variables**. En general, sin embargo, x_i podría ser un objeto estructurado complejo, como una imagen, una oración, un mensaje de correo electrónico, una serie de tiempo, una forma molecular, un gráfico, etc. De manera similar, la forma de la variable de salida *output* o respuesta puede ser en principio cualquier cosa, pero la mayoría de los métodos asumen que y_i es una **variable categórica o nominal** de un conjunto finito, $y_i \in 1, ..., C$ (como masculino o femenino), o que y_i es un valor real escalar (como el nivel de ingresos). Cuando y_i es categórico, el problema se conoce como **clasificación o reconocimiento de patrones**, y cuando y_i tiene un valor real, el problema se conoce como **regresión**. Otra variante, conocida como **regresión ordinal**, ocurre donde el espacio de la etiqueta y tiene algún orden natural, como los grados A a F [Murphy, 2012].

- El segundo tipo principal de aprendizaje automático es el enfoque de **aprendizaje descriptivo o no supervisado**. Aquí solo se dan entradas, $D = \{x_i\}_{i=1}^{N}$, y el objetivo es encontrar "patrones interesantes" en los datos. Esto a veces se llama **descubrimiento del conocimiento**. Este es un problema mucho menos definido, ya que no se nos dice qué tipos de patrones buscar, y no hay una métrica de error evidente que usar (a diferencia del aprendizaje supervisado, donde se puede comparar la predicción de y para cada x dado del valor observado) [Murphy, 2012].

- Hay un tercer tipo de aprendizaje automático, conocido como **aprendizaje por refuerzo**, que es menos utilizado. En un entorno de aprendizaje de refuerzo típico, un agente interactúa con su entorno y recibe una función de recompensa que trata de optimizar, por ejemplo, el sistema puede ser recompensado por ganar un juego. El objetivo del agente es conocer las consecuencias de sus decisiones, del ejemplo, qué movimientos fueron importantes para ganar un juego y usar este aprendizaje para encontrar estrategias que maximicen sus recompensas [Royal Society (Great Britain), 2017].

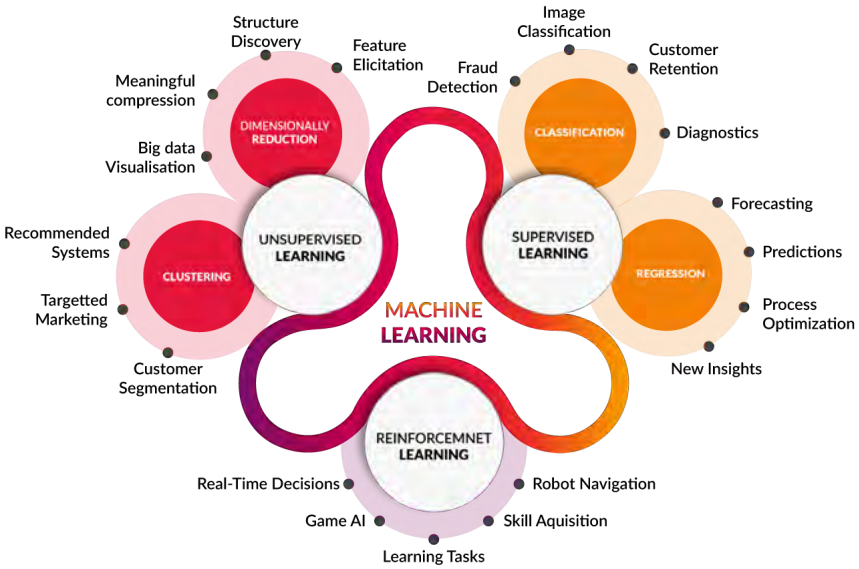


Figura 2.4 Tipos de aprendizaje automático
Fuente: https://bit.ly/2OexcoQ

2.2.2.2. Aprendizaje automático supervisado.

En los algoritmos de aprendizaje automático supervisado *(Supervised Learning)* se genera un modelo predictivo, basado en datos de entrada y salida. La palabra clave "supervisado" viene de la idea de tener un conjunto de datos previamente etiquetado y clasificado, es decir, tener un conjunto de muestra el cual ya se sabe a qué grupo, valor o categoría pertenecen los ejemplos. Con este grupo de datos, el cual llamamos datos de entrenamiento, se realiza el ajuste al modelo inicial planteado. De esta forma es como el algoritmo va "aprendiendo" a clasificar las muestras de entrada comparando el resultado del modelo, y la etiqueta real de la muestra, realizando las compensaciones respectivas al modelo de acuerdo a cada error en la estimación del resultado [Gonzalez, 2018].

Existen dos tipos de aprendizaje automático supervisado:

1. Clasificación *(Classification)*: en este tipo, el algoritmo encuentra diferentes patrones y tiene por objetivo clasificar los elementos en diferentes grupos.

2. Regresión *(Regression)*: La regresión es como la clasificación, excepto que la variable de respuesta es continua. Tenemos una única entrada de valor real $x_i \in \mathbb{R}$, y una única respuesta de valor real $y_i \in \mathbb{R}$. Se considera que se ajustan dos modelos a los datos: una línea recta y una función cuadrática. Surgen varias extensiones de este problema básico, como tener entradas de alta dimensión, valores atípicos, respuestas no

uniformes, etc [Murphy, 2012].

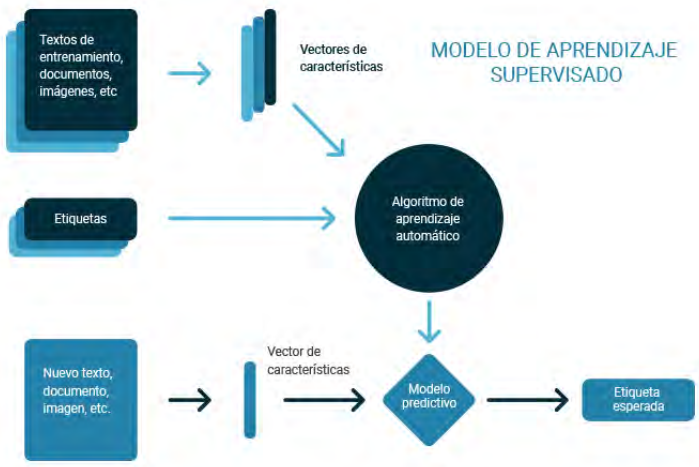


Figura 2.5 Diagrama de flujo del aprendizaje automático supervisado
Fuente: https://bit.ly/2C7ATWb

2.2.2.3. Clasificación.

Es hacer un mapeo de *input* x a sus *output* y, donde $y \in \{1 \dots C\}$, siendo C el número de clases. Si $C = 2$, se llama **clasificación binaria** (en cuyo caso a menudo asumimos $y \in \{0, 1\}$); Si $C > 2$, se llama **clasificación multiclase**. Si las etiquetas de clase no son mutuamente excluyentes (por ejemplo, alguien puede ser clasificado como alto y fuerte), lo llamamos **clasificación de múltiples etiquetas**, pero esto se ve mejor como predicción de múltiples etiquetas de clase binarias relacionadas (llamado **modelo de salida múltiple**). Cuando usamos el término "clasificación", se refiere a la clasificación multiclase con una sola salida, a menos que se indique lo contrario.

Un modo de formalizar el problema es una **función de aproximación**. Se asume que $y = f(x)$ para alguna función desconocida f, y el objetivo del aprendizaje es estimar la función f dado un conjunto de entrenamiento etiquetado, y luego hacer predicciones utilizando $\hat{y} = \hat{f}(x)$. (Se usa el símbolo del sombrero para indicar una estimación). El objetivo principal es hacer predicciones sobre nuevos *inputs*, es decir, la que no se ha visto antes (esto se llama **generalización**), ya que predecir la respuesta en el conjunto de entrenamiento es fácil (sólo es buscar la respuesta)[Murphy, 2012].

2.2.3. Artificial neural network.

En español **red neuronal artificial** o en su siglas en inglés **ANN**, surge del intento de simular los sistemas nerviosos biológicos combinando muchos elementos de computacion simples (neuronas) en un sistema altamente interconectado y esperando que fenomenos complejos como la "inteligencia" surjan como resultado de la auto-organización o el aprendizaje [Sarle, 1994].

Consiste en un conjunto de elementos simples de procesamiento llamados nodos o neuronas conectadas entre sí, por conexiones que tienen un valor numérico modificable llamado peso. Ver figura 2.6. La actividad que una unidad de procesamiento o neurona artificial realiza en un sistema de este tipo es simple. Normalmente, consiste en sumar los valores de las entradas (*inputs*) que recibe de otras unidades conectadas a ella, comparar esta cantidad con el valor umbral y, si lo iguala o supera, enviar activación o salida (*output*) a las unidades que esté conectada. Tanto las entradas que la unidad recibe como las salidas que envía dependen a su vez del peso o fuerza de las conexiones por las cuales se realizan dichas operaciones [Montaño Moreno, 2017].

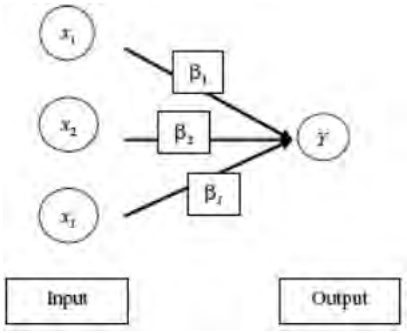


Figura 2.6 Red neuronal artificial simple
Fuente:https://bit.ly/2UDlUdQ

2.2.3.1. Arquitectura.

Se denomina arquitectura a la topología, estructura o patrón de conexiones de una red neuronal. Los nodos se conectan por medio de sinapsis, estando el comportamiento de la red determinado por la estructura de conexiones. Estas conexiones sinápticas son direccionales, es decir, la información solamente puede propagarse en un único sentido (desde la neurona presináptica a la pos-sináptica). En general las neuronas se suelen agrupar en unidades estructurales que denominaremos capas. El conjunto de una o más capas constituye la red neuronal.

Se distinguen tres tipos de capas: entrada, salida y ocultas. Una capa de entrada, también denominada sensorial, está compuesta por neuronas que reciben datos o señales procedentes del entorno. Una capa de salida se compone de neuronas que proporcionan la respuesta de la red neuronal. Una capa oculta no tiene una conexión directa con el entorno, es decir, no se conecta directamente ni a órganos sensores ni a efectores. Este tipo de capa oculta proporciona grados de libertad a la red neuronal gracias a los cuales es capaz de representar más fehacientemente determinadas características del entorno que trata de modelar. Figura 2.7. Teniendo en cuenta diversos conceptos se pueden establecer diferentes tipos de arquitecturas neuronales.
Considerando la estructura podemos hablar de redes:

- **Mono capa**: Compuestas por una única capa de neuronas.
- **Redes multicapa**: Las neuronas se organizan en varias capas.

Teniendo en cuenta el flujo de datos podemos distinguir:

- **Redes unidireccionales *(feedforward)***: la información circula en un único sentido.
- **Redes recurrentes o realimentadas (feedback)**: la información puede circular entre las distintas capas de neuronas en cualquier sentido, incluso en la de salida a entrada.

[Haykin, 1999]

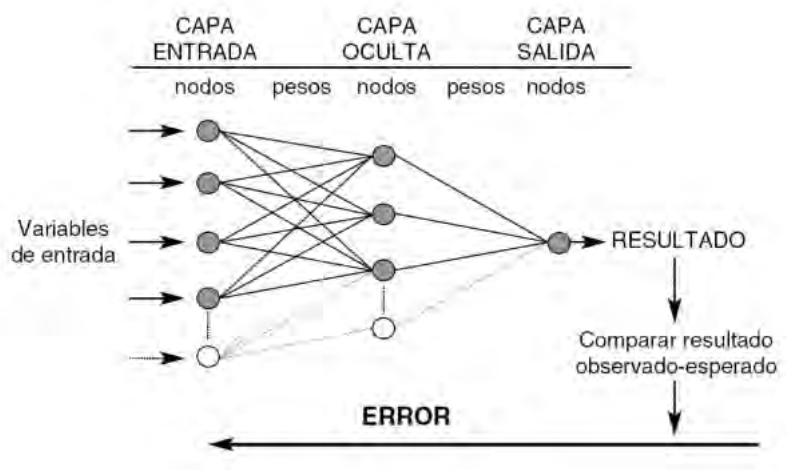


Figura 2.7 Arquitectura de una red neuronal artificial (perceptrón de tres capas).
Fuente:https://bit.ly/2QREQqg

2.2.4. Deep Learning

En los últimos años, se ha cambiado considerablemente la forma de trabajar en el procesamiento de señales [Deng and Yu, 2014]. Los avances en investigación de procesamiento de señales, big data y el drástico incremento de la potencia de cómputo en CPU y GPU han hecho mejoras sustanciales en las aplicaciones del deep learning para las tareas de visión de computador. Ejemplo de aplicaciones exitosas incluyen tareas de clasificación, localización y extracción de características supervisadas y no supervisada [Cireşan et al., 2012].

En cada nivel de deep learning se aprende a transformar los datos de entrada en una representación un poco más abstracta y compuesta. En una aplicación de reconocimiento de imágenes, la entrada sin formato puede ser una matriz de píxeles; la primera capa de representación puede abstraer los píxeles y codificar los bordes; la segunda capa puede componer y codificar arreglos de bordes; la tercera capa puede codificar una nariz y ojos; y la cuarta capa puede reconocer que la imagen contiene una cara. Es importante destacar que un proceso de aprendizaje profundo puede aprender qué características ubicar de manera óptima en qué nivel por sí solo . (Por supuesto, esto no elimina completamente la necesidad del ajuste manual; por ejemplo, un número variable de capas y tamaños de capas puede proporcionar diferentes grados de abstracción) [Deng and Yu, 2014]. Las arquitecturas de aprendizaje profundo a menudo se construyen con un método codicioso capa por capa. El aprendizaje profundo ayuda a desenredar abstracciones y elegir qué características mejoran el rendimiento [Bengio et al., 2013].

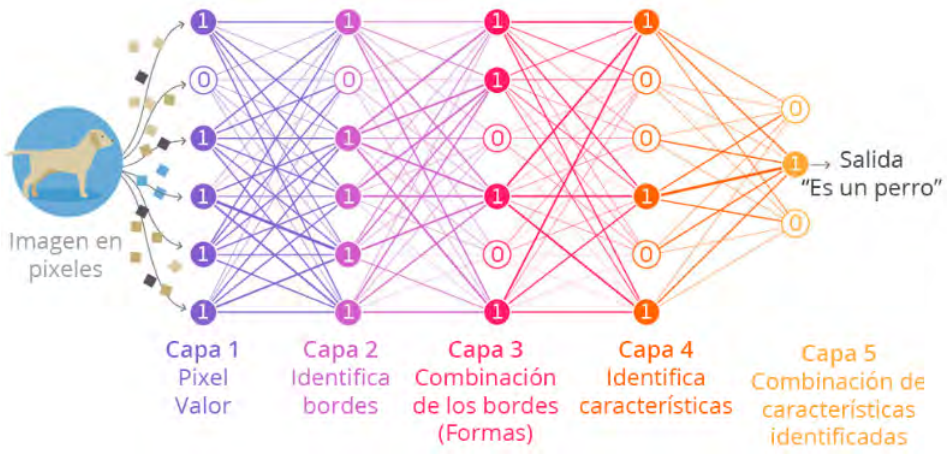


Figura 2.8 Deep Learning
Fuente:https://bit.ly/2EatGqV

2.2.4.1. *Técnicas Deep Learning.*

El deep learning se puede definir como redes neuronales con gran cantidad de parámetros y capas, las cuatro técnicas de redes principales son: [Le, 2017]

- ***Unsupervised Pre-trained Networks:*** El entrenamiento previo no supervisado a nivel de capas supera los desafíos del deep learning al introducir un ajuste fino antes del procedimiento de entrenamiento supervisado. Asegura que el efecto de regularización es una consecuencia del procedimiento de pre-entrenamiento que establece un punto de comercialización del procedimiento de ajuste fino dentro de una región de espacio de parámetros en los que los parámetros están restringidos en adelante. Los parámetros están restringidos a un volumen relativamente pequeño de espacio de parámetros que está delineado por el límite de cuenca local de atracción de la función supervisada de costo de ajuste fino. El procedimiento de pre-entrenamiento aumenta la magnitud de los pesos y en los modelos estándar profundos, con una no linealidad sigmoidal, esto tiene el efecto de hacer que la función sea no lineal y el costo local más complicado, con más características topológicas como picos, canales y mesetas. La existencia de estas hace que el espacio del parámetro sea más difícil a nivel local para recorrer distancias significativas a través de un procedimiento de descenso de gradiente. Este es el núcleo de la propiedad restrictiva impuesta por el procedimiento de pre-entrenamiento y, por lo tanto, la base de sus propiedades de regularización [Erhan et al., 2010].

- ***Convolutional Neural Networks:*** Las redes neuronales convolucionales son similares a las redes neuronales multicanal, su principal ventaja es que cada parte de la red se le entrena para realizar una tarea, esto reduce significativamente el número de capas ocultas, por lo que el entrenamiento es más rápido. Además, presenta invarianza a la traslación de los patrones a identificar [Calvo, 2017], es una red neuronal estándar que se ha extendido a través del espacio utilizando ponderaciones compartidas. La CNN está diseñada para reconocer imágenes al tener convoluciones dentro, que ven los bordes de un objeto reconocido en la imagen [Le, 2017]. La arquitectura más convencional de las CNN consiste en una o más capas convolucionales (que pueden estar conectadas a capas *pooling*), luego están conectadas a una o más capas totalmente conectadas *(fully-connected)*, que son las mismas que en las redes neuronales multicapa convencionales, y finalmente, a una capa de salida. Estas redes están diseñadas para aprovechar la estructura de su entrada, como por ejemplo imágenes. Esto se consigue gracias a que las propiedades (características) aprendidas en una parte de la imagen

pueden usarse para las otras partes. Otra ventaja de este tipo de redes es que son más fáciles de entrenar y tienen menos parámetros que una red neuronal multicapa con el mismo número de neuronas ocultas [Ortega and Andrés, 2018].

- ***Recurrent Neural Networks:*** Es una clase de red neuronal artificial donde las conexiones entre los nodos forman un gráfico dirigido a lo largo de una secuencia. Esto le permite exhibir un comportamiento dinámico temporal para una secuencia de tiempo. A diferencia de las redes neuronales de alimentación directa , los RNN pueden usar su estado interno (memoria) para procesar secuencias de entradas. Esto los hace aplicables a tareas como el reconocimiento de escritura a mano no segmentado, conectado o el reconocimiento de voz [Graves et al., 2009]. Está diseñado para reconocer secuencias, por ejemplo, una señal de voz o un texto. Tiene ciclos dentro que implican la presencia de memoria corta en la red [Le, 2017]. Es un modelo de secuencia neuronal que logra un rendimiento de vanguardia en tareas importantes que incluyen el modelado del lenguaje [Mikolov and Zweig, 2012].

- ***Recursive Neural Networks.*** Es un tipo de red neuronal profunda creada al aplicar el mismo conjunto de ponderaciones recursivamente sobre una entrada estructurada, para producir una predicción, estructuras de entrada de tamaño variable, o una predicción escalar en ella, al atravesar una estructura dada en orden topológico [Goller and Kuchler, 1996]. Es más como una red jerárquica donde realmente no hay un aspecto de tiempo para la secuencia de entrada, pero la entrada debe procesarse jerárquicamente en forma de árbol [Le, 2017]. Las redes neuronales recursivas comprenden una clase de arquitectura que puede operar con entradas estructuradas. Anteriormente, se han aplicado con éxito para modelar la composicionalidad en lenguaje natural utilizando representaciones estructurales basadas en árbol de análisis. A pesar de que estas arquitecturas son de estructura profunda, carecen de la capacidad de representación jerárquica que existe en las redes convencionales de avance profundo, así como en las redes neuronales recurrentes profundas recientemente investigadas [İrsoy and Cardie, 2014].

2.2.5. Convolutional Neural Networks.

También conocidas como redes neuronal convolucionales, o CNN por sus siglas en inglés *(Convolutional Neural Network)*, es un tipo especializado de red neuronal de aprendizaje supervisado que se utiliza para procesar data con características similares, destacando en tareas de visión por computadora, como la clasificación de imágenes. Las CNN se inspiraron en los procesos biológicos del patrón de conectividad entre las neuronas se inspira en la organización de la corteza visual animal [Galárraga Cañizares, 2017].

Es aquella red neuronal que entre la capa de entrada y de salida existen diversas capas. Estas capas reciben el nombre de capas ocultas *(hidden layers)*. La arquitectura más convencional de las CNN tiene una o más capas convolucionales (pueden estar conectadas a capas *pooling)*, luego están conectadas a una o más capas totalmente conectadas *(fully-connected)*, las mismas que en las redes neuronales multicapa convencionales y finalmente a una capa de salida.

Este tipo de red normalmente usa como entrada una imagen, es decir, $m * m * r$ valores donde m es el ancho y alto de la imagen y r el número de canales, por ejemplo en *RGB*, $r = 3$ y como salida devolverá un $i \in 0...K-1$ donde $K =$ número de clases. [Ortega and Andrés, 2018]

2.2.5.1. Componentes.

Capa convolucional.

La entrada a una capa convolucional es un array de (m, m, r) valores. Además, se tienen que pasar cuatro hiper-parámetros, estos son:

- **Número de filtros *(kernels)*:** Número de *feature maps* extraídos de la imagen. Un *feature map* es un mapa de características encontradas en la imagen. Como por ejemplo, curvas, líneas rectas. Los feature maps de capas más alejadas de la capa de entrada pueden procesar feature maps más complejas como por ejemplo, círculos, patrones de curvas, entre otras.

- **Tamaño filtro *(Kernel size)*:** Un filtro es una región de la imagen de un determinado tamaño.

- ***Stride:*** Específica en cuántos píxeles se desplazará el filtro.

- ***Padding****:* Indica el número a añadir a los márgenes de la entrada con el objetivo de modificar el tamaño de la salida de la capa.

[Ortega and Andrés, 2018]

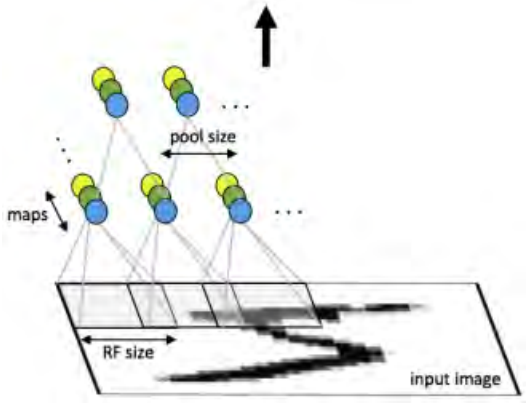


Figura 2.9 División de imágenes en distintos feature maps
Fuente: [Ortega and Andrés, 2018].

En esta capa se computan k f*eature maps*, como se puede observar en la figura 2.9 , que matemáticamente son expresados de la siguiente forma para el valor (i, j) en el k-ésimo feature map:

$$z_{i,j,k} = w_k^T x_{i,j} + b_k \tag{2.1}$$

donde w_k y b_k son los vectores de peso y el término independiente del k-ésimo filtro. Los valores $x_{i,j}$ son extraídos de la región que está tratando el filtro en ese momento y no de la imagen entera. Luego a este valor se le aplica una función de activación, como se puede observar en la figura 2.10 . El cometido es aportarle no linealidad, ya que es interesante detectar *features* que no presenten propiedades lineales. [Ortega and Andrés, 2018]

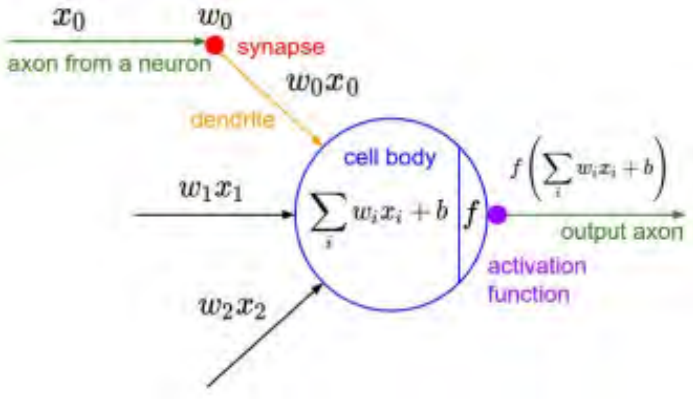


Figura 2.10 Entrada y salida de una neurona usando función de activación
Fuente: [Ortega and Andrés, 2018].

La función ReLU consiste en la función $f(x) = max(0, x)$ la gráfica que se muestra en

la figura 2.11. Básicamente, esta función sirve como umbral de los números negativos. Se ha comprobado que el entrenamiento de una red CNN con ReLU es mucho más rápido. Además, esta función presenta una gran simplicidad respecto a las funciones sigmoide y tanh, ya que no contiene operaciones como el exponencial. [Ortega and Andrés, 2018]

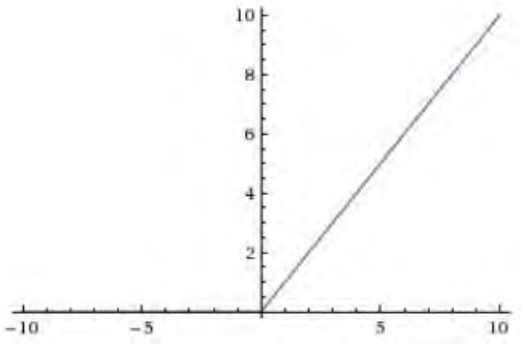


Figura 2.11 Comportamiento de la funcion ReLU
Fuente: [Ortega and Andrés, 2018].

La función sigmoide tiene la siguiente forma:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.2}$$

Esta función recibe como entrada un número real y obtiene como salida un número en el rango $[0..,1]$. En particular, los números con un gran valor negativo se convierten en 0 y los números con un gran valor positivo se convierten en 1. Como se puede observar en la figura 2.12. [Ortega and Andrés, 2018]

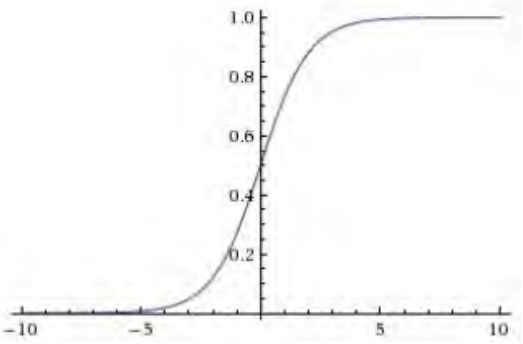


Figura 2.12 Comportamiento de la función Sigmoide
Fuente: [Ortega and Andrés, 2018].

La función tanh tiene la siguiente forma:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

Esta función recibe como entrada un número natural y obtiene como salida un número en el rango $[-1..,1]$. Como se puede observar en la figura 2.13. Esta función es más usada que la función sigmoide, ya que su salida está centrada en el 0, presenta propiedades deseables a la hora de ejecutar el algoritmo de Back-propagation [Ortega and Andrés, 2018].

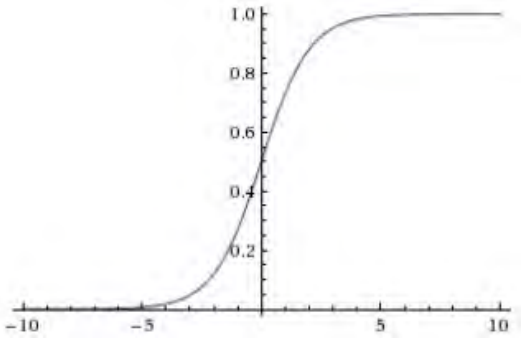


Figura 2.13 Comportamiento de la función tanh
Fuente: [Ortega and Andrés, 2018].

Capa Pooling.

Es una capa que usualmente está situada entre dos capas convolucionales cuya función es reducir el tamaño espacial de su entrada y así reducir la cantidad de parámetros y computación en la red, como se puede observar en la figura 2.14. Una de las consecuencias de esta reducción, es el control del *overfitting*, es decir, que la red neuronal aprenda a diferenciar entre las clases de las imágenes con las que ha sido entrenado y no generalice a aquellas que no ha visto. Esta capa toma como hiper-parámetros: El tamaño del filtro al que aplicará su operación y stride. Este último funciona igual que en la capa convolucional. Las funciones más comunes de *pooling* son *max-pooling* y *average-pooling*. Estas realizan la función que su nombre indica. *Max-pooling* toma la entrada la región especificada en el hiper-parámetro tamaño del filtro y le aplica la función max. *Average-pooling* tiene el mismo comportamiento que *max-pooling*, pero con la diferencia de que aplica la función *average* [Ortega and Andrés, 2018] .

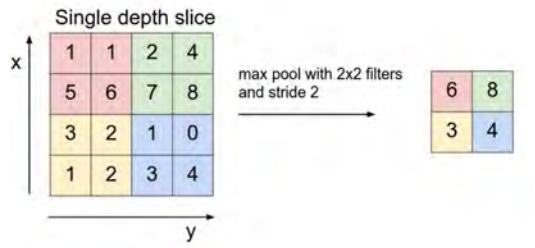


Figura 2.14 Ejemplo de max-pooling
Fuente: [Ortega and Andrés, 2018].

Capa *Fully-connected*.

Esta capa es la encargada de computar la misma función de la capa convolucional, pero con la diferencia de que la capa convolucional estaba conectada localmente a los valores correspondientes al tamaño de su filtro, en cambio, esta capa está totalmente conectada a todas las neuronas de capa anterior. En la figura 2.15 se puede observar como cada una de las neuronas de cada capa está conectada a cada una de las neuronas de la capa anterior [Ortega and Andrés, 2018].

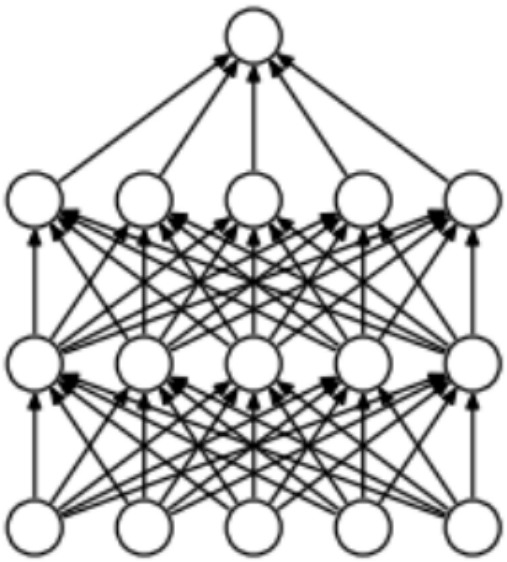


Figura 2.15 Ejemplo de capas fully-connected
Fuente: [Ortega and Andrés, 2018].

Capa de Salida.

Para funciones de clasificación la última capa de la red, es decir, la capa de salida es una capa *fully-connected* que tiene como función de activación usualmente *softmax*. La función *softmax* genera como resultado una distribución de probabilidades sobre las clases. Esta función es de la siguiente forma:

$$\sigma(Z_j) = \frac{e^{Z_j}}{\sum_{k=1}^{K} e^{Z_k}} \tag{2.4}$$

donde K es el número de clases, $j \in 1...KyZ_j$ es el vector de entrada de la neurona j.

Al usar *softmax* implícitamente se utiliza la función de *loss* (ver sección Función de *loss*) llamada *categorical cross-entropy* que toma como entrada las probabilidades calculadas por *softmax*. Esta función tiene la siguiente forma:

$$E = -\sum_{k=1}^{k} t_k \log y_k \tag{2.5}$$

donde el vector t es la clase de la imagen en la forma $(0,0,..,0,1,0.,0)$ donde solo puede ser 1 el elemento que representa la $i-th$ clase, y es el vector resultante de la función *softmax* en la misma forma que t.

[Ortega and Andrés, 2018]

Función *Loss*.

Esta red neuronal es entrenada usando el algoritmo de *Back-propagation*. Este algoritmo contiene dos fases, la fase de f*orward-propagation* y la fase de *back-propagation*. La primera fase consiste en administrar un *input* a la red en forma de (x,t) donde la x es la imagen y la t la clase a la que pertenece la imagen, y ésta hará el cómputo capa por capa hasta llegar a la capa de salida. El valor de la capa de salida es comparado a t usando la función *loss*. Después, el valor del error es calculado para cada una de las neuronas de la capa de salida y es propagado hacia las capas anteriores hasta llegar a la primera capa (capa de entrada). Así, cada una de las neuronas tiene un valor de error que intuitivamente puede verse como que tan responsable es la neurona del valor de salida. En la segunda fase, se computa los gradientes para cada neurona usando los errores calculados anteriormente y los pesos. Más tarde, se usa un optimizador para actualizar los pesos de las neuronas basándose en los gradientes con el objetivo de minimizar la función *loss*. Los optimizadores más comunes son SGD *(Stochastic Gradient Descendent)*, rmsprop y adam [Ortega and Andrés, 2018].

Las redes neuronales se entrenan mediante *epochs*, es decir, se pasa como input el conjunto de entrenamiento un número fijo de veces y a cada *epoch* se evalúa el *loss* que resulta usando una entrada que no haya visto la red. Esta entrada se conoce como conjunto de validación y este *loss* se conoce como *validation loss*, para diferenciarlo del *training loss* que corresponde al *loss* del conjunto de entrenamiento. Esta evaluación con el *validation loss* se realiza con el objetivo de determinar la capacidad de generalización de la CNN. Uno de los resultados que ofrece una red neuronal es *accuracy*, es decir, que porcentaje de los ejemplos vistos fue capaz de acertar. Cuando se refiere al conjunto de entrenamiento, se habla de *training accuracy* y cuando se refiere al conjunto de validación se refiere a *validation accuracy*.

[Ortega and Andrés, 2018]

Regularización.

Las redes neuronales tienden a hacer *overfitting*, es decir, que la red aprenda a diferenciar entre las clases de imágenes con las que ha sido entrenado y no generalice a aquellas que no ha visto. Para evitarlo se usan técnicas de regularización cuyo objetivo es prevenirlo. Las técnicas más usadas son: Aumentar el tamaño del *dataset*, *early-stopping*, regularización L1, L2 y *dropout*. La técnica dropout consiste en anular neuronas. Esto se refiere a que temporalmente se suspende de la red junto con sus conexiones de entrada y salida. La elección de que unidades anular es aleatoria y viene dada por el parámetro *ratio* [Ortega and Andrés, 2018].

2.2.6. Transfer Learning.

La transferencia de aprendizaje o *transfer learning*, se refiere a cualquier proceso algorítmico mediante el cual se utiliza la estructura o el conocimiento derivado de un problema de aprendizaje, para mejorar un problema diferente pero relacionado.[West et al., 2007]. La transferencia de aprendizaje es una técnica de aprendizaje automático, que consiste en entrenar una arquitectura neuronal para una determinada tarea y que esta arquitectura puede ser reutilizada para otra tarea similar. [Yosinski et al., 2014] Con el uso de esta técnica de aprendizaje, el proceso de aprendizaje no inicia desde cero, comienza a partir de patrones que han aprendido a resolver un problema diferente. Se utiliza comúnmente los modelos pre-entrenados para realizar el aprendizaje por transferencia.

2.2.6.1. Modelos pre-entrenados.

Consiste en utilizar un modelo pre-entrenado, y reemplazar sus últimas capas por otras, para que de esa manera se extraigan las características del nuevo conjunto de datos[Yaranga et al., 2018].

El reentrenamiento implica agregar sucesivamente una nueva capa oculta a un modelo y volver a montarlo, permitiendo que el modelo recién agregado aprenda las entradas de la capa oculta existente, a menudo mientras se mantienen fijos los pesos de las capas ocultas existentes [Goodfellow et al., 2016].

Beneficios.

Los beneficios clave para el uso de pre-entrenamiento son:

- Proceso de información simplificado.
- Facilidad en el desarrollo de redes más profundas.
- Útil como esquema de pesos.
- Menor error en la generalización.

  En general, el entrenamiento previo puede ayudar tanto en términos de optimización como en temimos de generalización [Goodfellow et al., 2016].

2.2.7. Persea Americana.

La Persea Americana comúnmente conocida como palta, aguacate, cura, avocado o abacate; es el fruto de un árbol originario de Centroamérica perteneciente a la familia de las lauráceas. Este árbol puede alcanzar una altura de entre diez y quince metros con un color de corteza gris-verdoso. Se clasifica en tres subespecies o razas ecológicas: americana, guatemalensis y drymifolia; son tres razas ecológicas que se desarrollaron por todo el continente americano. Se diferencian en la altura de la planta, en la forma y tamaño del fruto, color de follaje y adaptación a diferentes condiciones climáticas y suelo.

Esta expansión ha dado origen a una gran variedad de paltas pero pocas son aptas para el mercado local o de exportación; las más conocidas son Fuerte, Hass y Nabal, que se comercializan todo el año con marcada demanda y variada estacionalidad de producción. [Minagri, 2015]

2.2.7.1. Variedad Hass.

Esta variedad fue desarrollada en 1926 y patentada en 1935 por Rudolph G. Hass, en Habra Heights (California) es actualmente la más comercial del mundo, en virtud de la calidad de sus frutos, alto rendimiento en producción y maduración tardía. Pertenece a la raza guatemalteca Persea var. guatemalensis y se adapta a condiciones subtropicales, temperaturas de 5°C a 19 °C y alturas entre los 1.800 y 2.000 msnm. [Minagri, 2015]

Taxonomía [Minagri, 2017]

- **Familia:** Lauráceas
- **Género:** Persea
- **Subgénero:** Persea
- **Especie:** Persea Americana.

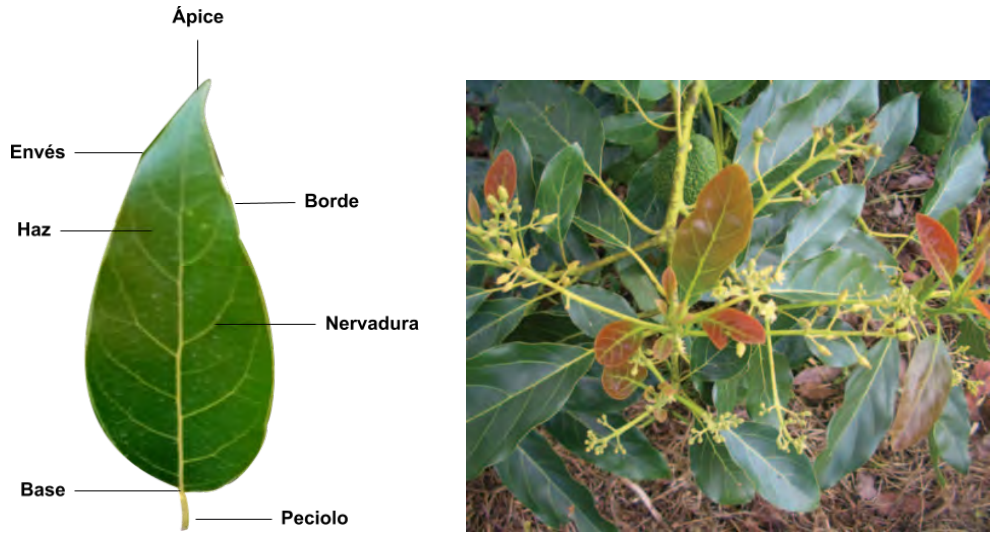
Morfología La Persea Americana Hass (principal variedad exportación) es una especie perenne, muy vigorosa, de crecimiento erecto; tiene forma oval piriforme, tamaño medio (200 a 300 gr.), piel gruesa, rugosa, se pela con facilidad y presenta color verde a oscuro violáceo cuando el fruto madura. La pulpa no tiene fibra y su contenido de aceite fluctúa entre 18 % y 22 %. La semilla es pequeña, de forma esférica y adherida a la pulpa. El fruto puede permanecer en el árbol un cierto tiempo después de alcanzar la madurez, sin perder su calidad. [Minagri, 2017]

Figura 2.16 Persea Americana
Fuente: https://bit.ly/2J2NKMD

Hoja Están dispuestas de forma alterna. Son pedunculadas, muy brillantes, de forma lanceolada, con base aguda, margen entero y ápice agudo (Figura 2.17(a)). El color de

las hojas maduras es verde mate, el pecíolo presenta estrías o surcos y el relieve de la venación por el haz es intermedio, usualmente levantado [Ríos et al., 2005].

Flor Es de color amarillo verdoso y densamente pubescente. Cada árbol puede llegar a producir hasta un millón de flores y el 0,1 % se transforma en fruto (Figura 2.17(b)). Las evaluaciones realizadas por [Ríos et al., 2005] muestran que la primera floración se presenta al año y medio.



(a) Partes de la hoja (b) Flores

Figura 2.17 Hojas y flores
Fuente: Elaboración propia.

Fruto Es ovalado, de tamaño pequeño a mediano, tiene corteza gruesa con textura de corcho y superficie áspera. Presenta un color verde oscuro cuando está en el árbol (Figura 2.18); cuando madura, toma un color verde púrpura. La semilla tiene un tamaño mediano y es redondeada; a su vez, la pulpa, a mediados y finales del proceso de maduración, es de color crema amarilla. Cuando el fruto ha alcanzado madurez fisiológica en zonas de clima frío, se puede dejar en el árbol por más tiempo. [Ríos et al., 2005]

Figura 2.18 Frutos
Fuente: Elaboración propia.

Tronco La superficie del tronco es rugosa, su ramificación es intensiva y la distribución de las ramas es verticulada. El color de las ramas jóvenes es rojo cobrizo, más intenso hacia el ápice. La superficie es pubescente y presenta lenticelas de color verde. [Ríos et al., 2005]

Figura 2.19 Tronco
Fuente: Elaboración propia.

2.2.7.2. Sintomatología de nutrientes.

Los requerimientos nutricionales de la Persea Americana son variables durante su desarrollo esto depende de la edad del árbol, fenología y cultivo por lo tanto la cantidad de nutrientes a aplicar está en función a la extracción de la planta, riqueza del suelo y aporte del agua de riego. El aporte de macroelementos deberá ser superior a los microelementos;

normalmente, los macroelementos se aplican al suelo, mientras los microelementos se aplican vía foliar. Los elementos nutritivos que las plantas necesitan se dividen en dos tipos, macroelementos y microelementos[1].[Whiley et al., 2013]

Macroelementos

a) **Nitrógeno (N):** Las plantas con gran producción de frutos y brotes nuevos requieren de un aporte adecuado de nitrógeno. El exceso en la aplicación de nitrógeno es un costo innecesario y contribuye a la contaminación por acumulación de nitratos, salinización o acidificación del suelo, puede causar la muerte de la planta. Los síntomas se presentan en hojas de color verde pálido, las nervaduras pueden tornarse amarillas. Senescencia temprana. Entre nudos cortos [Sigueñas., 2010]. Figura 2.20(a)

b) **Fósforo (P):** Durante el periodo de formación de la planta, el fósforo es el nutriente más importante por influir en el crecimiento de la raíz; también tiene un efecto significativo en el desarrollo de las flores y en el cuajado de los frutos. Se debe considerar que el suelo puede fijar el fósforo, no siempre hay un patrón definido para establecer esta deficiencia. síntomas principalmente en hojas maduras que puede tomar un color marrón, lucen más pequeñas y redondeadas. Pueden presentarse senescencia temprana, reducción en crecimiento e incluso muerte descendente [Sigueñas., 2010]. Figura 2.20(b)

c) **Potasio (K):** Después del nitrógeno, el potasio es el elemento más importante en el crecimiento y en la producción, ya que juega un papel significativo en los procesos de fotosíntesis, respiración y circulación de la savia. Síntomas principalmente en hojas maduras que pueden presentar clorosis intervenal, manchas de color rojo marrón, reducción en el tamaño. Las ramas pueden ser muy delgadas. Los síntomas de deficiencia aparecen primero en la base hojas y en pecíolos, entonces avanza a lo largo de la nervadura central. En el caso del aguacate Hass se debe tener cuidado con la aplicación de potasio por su efecto sobre el tamaño del fruto [Sigueñas., 2010]. Figura 2.20(c)

d) **Calcio (Ca):** Los síntomas de deficiencia de calcio se presentan en hojas maduras, puede presentarse reducción en su tamaño y quemazón en los bordes. Los síntomas pueden ser similares a los ocasionados por Fósforo. Una adecuada concentración de calcio en el fruto permite mantener la calidad en poscosecha [Sigueñas., 2010]. Figura 2.20(d)

[1]Más informacion de la sintomatología de microelementos en *" Buenas Prácticas Agrícolas en el Cultivo de Palto " de* [Sigueñas., 2010]

e) **Magnesio (Mg):** Este elemento es considerado como secundario, es un fuerte activador de enzimas. Los síntomas de su deficiencia aparecen en las hojas más viejas, asimismo se reduce el crecimiento y presenta una defoliación prematura. Las pocas hojas que quedan presentan una clorosis intervenal amarillo bronceado que avanza del borde al interior de la hoja [Sigueñas., 2010]. Figura 2.20(e)

f) **Azufre (S):** Este elemento participa en diversos procesos fisiológicos y su deficiencia altera el crecimiento de la planta. Los síntomas se presentan en las hojas jóvenes son más susceptibles que las maduras. Pueden presentar amarillamiento y reducción en su tamaño [Sigueñas., 2010]. Figura 2.20(f)

(a) Nitrógeno

(b) Fosforo

(c) Potasio

(d) Calcio

(e) Magnesio

(f) Azufre

Figura 2.20 Deficiencia de Macro elementos
Fuente: Elaboración propia.

2.3. Software empleado en el desarrollo

2.3.1. Tecnologías

2.3.1.1. *Python.*

Python es un lenguaje de programación multiparadigma, que significa que combina propiedades de diferentes paradigmas de programación orientado a objetos también incorpora aspectos de programación imperativa y funcional. Es interpretado en el tiempo de ejecución, además es de tipado dinámico porque permite la mutación de variables y multiplataforma [pyt, 2019].

2.3.1.2. *TensorFlow.*

TensorFlow [ten, 2019] es una biblioteca de software de código abierto para el cálculo numérico que utiliza grafos de flujo de datos. Los nodos en el grafo representan operaciones matemáticas, mientras que las aristas del grafo representan los conjuntos de datos multidimensionales (tensores) comunicados entre ellos. La arquitectura flexible le permite realizar cálculos en una o más CPU o GPU de un ordenador, servidor o dispositivo móvil desde la misma API. TensorFlow fue desarrollado originalmente por investigadores e ingenieros que trabajan en el equipo Brain de Google dentro de la organización de investigación de Inteligencia Artificial para realizar investigaciones de aprendizaje automático y redes neuronales profundas, pero la plataforma es lo suficientemente general como para aplicarse en una amplia variedad de dominios. Tensorflow fue puesto como un recurso *open-source* en Noviembre del 2015.

2.3.1.3. *Keras.*

Keras [ker, 2019] es una API de redes neuronales escrita en Python. Es un *wrapper* capaz de ejecutarse encima de TensorFlow, CNTK, or Theano. Específicamente se usará en conjunto con Tensorflow. Soporta las redes neuronales convolucionales y las redes recurrentes, además de permitir la ejecución tanto en CPU como en GPU.

Los objetivos que persigue el framework de Keras son principalmente: ser una plataforma *user-friendly* y de fácil extensibilidad. Además, tiene un gran acoplamiento con Python, ya que soporta las versiones 2.7 y 3.6.

2.3.1.4. *Flask.*

Flask es un *framework* minimalista escrito en Python que permite crear aplicaciones web [fla, 2019a].

2.3.1.5. *Flask-RESTful.*

Es una extensión para *Flask* que agrega soporte para la creación rápida de API REST. Es una abstracción ligera que funciona con ORM [fla, 2019c].

2.3.1.6. *Flask-Cors.*

Una extensión de Flask para el manejo del intercambio de recursos de origen cruzado (CORS), que hace posible el AJAX de origen cruzado [fla, 2019b].

2.3.1.7. *Opencv-Python.*

Es una librería adecuada para la creación rápida de prototipos de problemas de visión de computadora [ope, 2019].

2.3.1.8. *Numpy.*

Es el paquete fundamental para la computación científica con Python [num, 2019].

2.3.1.9. *Framework Angular 5.*

Es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles [ang, 2019].

2.3.1.10. *Bootstrap.*

Es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales [boo, 2019].

2.3.1.11. *jQuery.*

Es una biblioteca multiplataforma de *JavaScript*, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica *AJAX* en páginas web [Foundation, 2019].

2.3.1.12. *MongoDB.*

MongoDB [mon, 2019b] es una base de datos NoSQL. Mongo DB es distribuida, por lo que es fácil de usar y proporciona una elevada disponibilidad, escalabilidad horizontal y distribución geográfica. Almacena datos en documentos JSON flexibles, es decir, cada documento puede contener diferentes campos y las estructuras de datos se pueden ir modificando.

2.3.1.13. *Mongoengine.*

Es un Mapeador de objetos y documentos, escrito en Python para trabajar con MongoDB [mon, 2019a].

2.3.2. Herramientas

2.3.2.1. *Jupiter Notebook.*

Jupyter Notebook [jup, 2019], es un entorno de trabajo interactivo que permite desarrollar código de manera dinámica, a la vez que integran en un mismo documento tanto bloques de código, texto, gráficas o imágenes. Es un SaaS utilizado ampliamente en análisis numérico, estadística y machine learning, entre otros campos de la informática y las matemáticas.

2.3.2.2. *Google Colaboratory.*

Google Colaboratory, Google Colab o simplemente **Colab** [Google, 2019], es un servicio en la nube gratuito basado en Jupiter Notebook brinda GPU **gratuitas**. Tiene preinstaladas las librerías que son comúnmente utilizadas en el desarrollo de aplicaciones de aprendizaje profundo como Keras, Tensorflow, OpenCV y la posibilidad de instalar las librerías que el usuario requiera. Soporta versiones de Python 2.7 y 3.6. Según la disponibilidad de los servidores, los recursos de los entornos virtuales de Colaboratory son GPU Tesla K80,

alcanza hasta 12.73 Gb RAM y en cuanto a la capacidad del disco entre 23.56 Gb hasta 358.27 Gb.

2.3.2.3. *Git.*

Git [git, a] es un sistema de control de versiones distribuido gratuito y de código abierto diseñado para manejar todo, desde proyectos pequeños hasta muy grandes, con rapidez y eficiencia. Es una herramienta para desarrollo de trabajos de software en equipo que permite la gestión eficiente de proyectos.

2.3.2.4. *GitLab.*

GitLab [git, b] comenzó como un proyecto de código abierto para ayudar a los equipos a colaborar en el desarrollo de *software*. GitLab ahora proporciona una aplicación única para todo el ciclo de vida de desarrollo de software y operaciones. GitLab proporciona todo lo que necesita para administrar, planificar, crear, verificar, empaquetar, lanzar, configurar, monitorear y proteger sus aplicaciones.

Capítulo 3

Metodología

3.1. Nivel y tipo de investigación

3.1.1. Nivel de investigación.

El nivel de la investigación es descriptiva/exploratoria[1], porque detalla la sintomatología ocasionadas por la deficiencia de macronutrientes en la Persea Americana variedad Hass y construye un conjunto de datos para identificar el macronutriente que origina la anomalía mediante arquitecturas de redes convolucionales.

3.1.2. Tipo de investigación.

El tipo de investigación es descriptiva/exploratoria porque detalla las deficiencias de los macronutrientes en la persea americana y construye un conjunto de datos. Además es de tipo tecnológica/aplicada porque materializa los resultados obtenidos en el desarrollo de un prototipo para un usuario final.

3.2. Unidad de estudio

Deficiencia de macronutrientes: Los macronutrientes son aquellos elementos que las plantas requieren en mayores cantidades para su crecimiento y provechosa producción de frutos. En consecuencia, la deficiencia de macronutrientes puede perjudicar desde el crecimiento, la decoloración de hojas hasta la pérdida de frutos.

[1]Dentro de la clasificación presentada por Roberto Hernández Sampieri [Sampieri, 2018]

3.3. Diseño metodológico

El trabajo de investigación no cuenta con hipótesis; el desarrollo de la misma fue orientada a los objetivos planteados en la sección 1.2. El diseño metodológico es experimental pues los objetivos formulados son correlacionales; lo que significa que se realizaron pruebas para alcanzar el objetivo general. Los datos utilizados fueron recolectados en el sector: Molinopata, provincia: Abancay, región: Apurimac.

3.4. Pasos de diseño metodológico

- **Determinar características en las hojas afectadas por deficiencia de macronutrientes.** Se determinaron las características más relevantes que presenta las hojas de la Persea Americana afectadas por la deficiencia de macronutrientes, se utilizó como referencia el Manual técnico de Buenas Practicas Agrícolas en el Cultivo de Palto desarrollado por el Ministerio de Agricultura del Perú.

- **Recolectar imágenes y agrupar por macronutriente.** En esta etapa se tomaron imágenes de hojas afectadas por la deficiencia de macronutrientes, en el sector: Molinopata, provincia: Abancay, región: Apurimac. Se utilizaron diferentes cámaras y fueron agrupadas según las características determinadas en la etapa anterior.

- **Elaborar *dataset* con las imágenes recolectadas en campo.** Una vez finalizada la recolección y el agrupamiento de imágenes, se procedió a un preprocesamiento que comprende las fases: depurar, dimensionar, normalizar y agrupar por categorías, para ser divididas en dos subconjuntos de entrenamiento y validación.

- **Analizar arquitecturas y características de redes convolucionales.** En esta etapa de investigación se analizaron las arquitecturas, características, tendencias en la clasificación de imágenes y trabajos previos que ayudaron a solucionar el problema de investigación.

- **Definir estrategia de aprendizaje**. Según las características del *dataset* se definió la estrategia de aprendizaje que fue empleada para los modelos de redes neuronales convolucionales.

- **Ajustar parámetros de entrenamiento de los modelos elegidos.** Se configuraron los parámetros de entrenamiento según las características de cada modelo seleccionado.

- **Entrenar los modelos de predicción.** Configurado los parámetros, se procedio a entrenar los modelos de redes neuronales convolucionales utilizando las máquinas virtuales que proporciona el proyecto *Google Colab*.[2]

- **Elaborar el prototipo para visualización de resultados.** Se elaboró un prototipo que permita visualizar los resultados obtenidos en el proceso de predicción, fueron mostrados mediante un diagrama de barras y gráficos circulares con los porcentajes obtenidos.

- **Evaluación y análisis de resultados.** Fueron comparados los resultados de entrenamiento y predicción de las arquitecturas seleccionadas, utilizando matrices de confusión y métricas de evaluación que permitieron medir el desempeño en la identificación de anomalías presentes en las hojas de Persea Americana causadas por la deficiencia de macronutrientes.

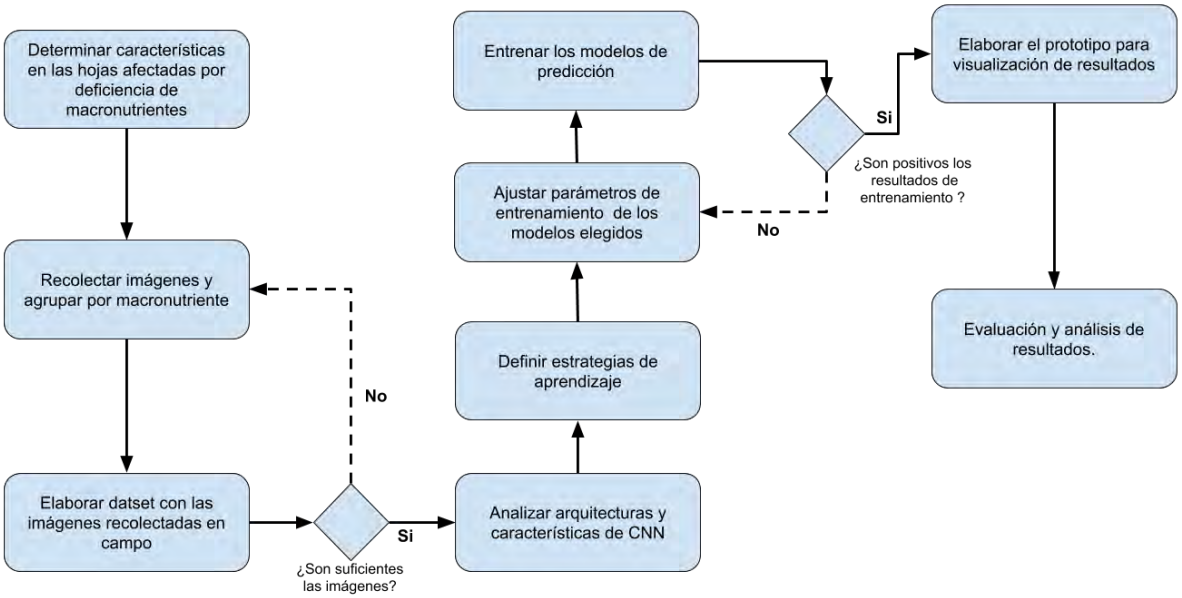


Figura 3.1 Pasos del Diseño metodológico
Fuente: Elaboración propia

[2]Colab es un servicio cloud de maquias virtuales de uso gratuito.[Google, 2019]

Capítulo 4

Desarrollo del Proyecto

4.1. Detalles Técnicos

Durante el desarrollo del proyecto, se utilizó recursos de *hardware* y *software*, que se detallan a continuación.

4.1.1. Hardware.

- **Cámara:** Sony Nex - C3, Lente: Sony N50.
- **Cámara:** Nikon Coolpix S6100.
- **Cámara:** Xiaomi Redmi Note 4.
- **Cámara:** Moto G6 Play.
- **Notebook:** Dell Inspiron 3458. Procesador Intel i5 5ta generacion 2.20 GHz, Ram 8 Gb DDR3, GPU: NVIDIA GeForce 820M 2 Gb.
- **Notebook:** Dell Inspiron 3458. Procesador Intel i5 5ta generacion 2.20 GHz, Ram 8 Gb DDR3, GPU: NVIDIA GeForce 820M 2 Gb.

4.1.2. Software.

- **Sistema operativo:** Deepin 15.10 x86_64 Linux.
- **Lenguaje de programación:** Python 3.6.5.
- **Framework backend:** Flask 1.0.2.

- **Gestor de base de datos:** MongoDB 3.4.14.
- **Framework frontend:** Angular 5.
- **Entorno de maquinas virtuales:** Google Colaboratory entorno de Jupyter Notebook, GPU Tesla K80, 12.73 Gb Ram, Disco de 23.56 Gb hasta 358.27 Gb.
- **Librerias:** Keras 2.2.4, Tensorflow 1.13.1, Numpy 1.16.2, Opencv-python 4.0.0.21, Matploitlib 1.5.1, Pandas 0.23.4.
- **Controlador de versiones:** Git 2.17.

4.2. Persea Americana Hass Dataset

El *dataset* elaborado para este trabajo de investigación lleva por nombre **Persea Americana Hass Dataset** [1]; contiene un total de 6,090 imágenes que corresponden a las hojas afectadas por deficiencia de macronutrientes, fueron recolectadas en el sector de Molinopata, provincia de Abancay, región de Apurímac.

(a) Sector (b) Recolección

Figura 4.1 Sector de recolección Molinopata
Fuente: Elaboración propia.

[1]El dataset elaborado se encuentra en el repositorio público de GitLab accesible por el siguiente enlace: *https://gitlab.com/betzabe/dshass.*

4.2.1. Sintomatología de las hojas.

Para la recolección de imágenes se identificó las características visibles en la hoja, en base a la tabla 4.1

Tabla 4.1 *Sintomatología de deficiencias de macronutrientes.*

Macronutriente	Síntomas
Calcio (Ca)	Síntoma principalmente en hojas maduras, necrosis en el ápice y quemazón en los bordes.
Potasio (K)	Síntoma principalmente en hojas maduras, presencia de mancha necróticas de color marrón rojizo en la base para luego diseminarse por toda la lámina de la hoja entre las nervaduras principales.
Magnesio (Mg)	Se observa clorosis intervenal en las hojas también color verde amarillo en el haz.
Nitrógeno (N)	Hojas de color verde pálido, las nervaduras pueden tornarse amarillas.
Fósforo (P)	Síntomas principalmente en hojas maduras que pueden tomar un color marrón rojizo y lucen más pequeñas y redondeadas.
Azufre (S)	Las hojas jóvenes son más susceptibles que las maduras. Pueden presentar amarillamiento, reducción en su tamaño y necrosis en nervadura.

Fuente: Sintetizado a partir de [Sigueñas., 2010].

Figura 4.2 Deficiencia de nutrientes
Fuente: Elaboración propia.

4.2.2. Recolección.

Las imágenes que corresponde a **Persea Americana Hass Dataset** fueron capturadas en el sector de Molinopata, provincia: Abancay, región: Apurímac; en cuatro oportunidades detalladas en la tabla 4.2

Tabla 4.2 *Periodo de recolección de imágenes.*

N°	Periodo	Observación
1	09/10/2018 - 13/10/2018	La captura de imágenes durante este periodo se realizó con un fondo blanco y luz artificial. Obteniendo un total de 970 imágenes
2	28/01/2019 - 01/02/2019	Durante este periodo se obtuvieron un total 2,850 imágenes, sin fondo blanco, se capturó tres imágenes por cada hoja en diferentes posiciones, distancias, con o sin presencia de objetos. Las imágenes fueron tomadas con luz natural.
3	04/02/2019 - 09/02/2019	Durante este periodo se obtuvieron un total 2,650 imágenes, sin fondo blanco, se tomó tres imágenes por cada hoja en diferentes posiciones, con o sin presencia de objetos. Las imágenes fueron tomadas con luz natural y artificial.
4	22/03/2019 - 31/03/2019	Durante este periodo se obtuvieron un total de 420, se tomó una imagen por cada hoja. Las imágenes fueron tomadas con luz natural y artificial.

Fuente: Elaboración propia

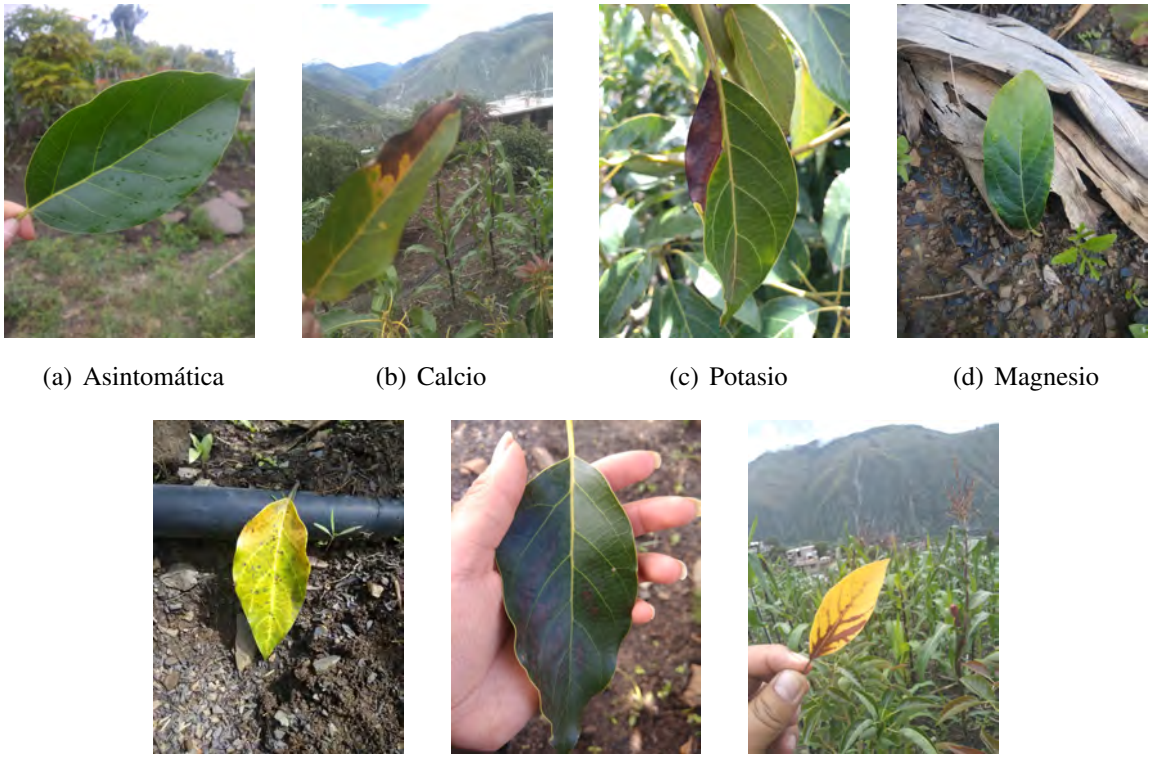
En el primer periodo de la recolección de imágenes presento más errores debido a que se capturaron cinco imágenes por cada hoja, sin ninguna modificación en la posición, distancia ni presencia de objetos. Al profundizar en el marco teórico del proyecto se concluyó que las imágenes capturadas en esta primera etapa podrían causar problemas de sobreajuste [Xu et al., 2019], es decir que el modelo de red neuronal memorizaría en lugar de aprender. Por este motivo se decide quitar el fondo blanco y agregar presencia de objetos en las imágenes.

Se recolectaron un total 6,890 imágenes, se depuraron imágenes: desenfocadas, dema-

siado iluminadas, muy oscuras, distorsionadas, entre otras, que generaron problemas para distinguir los rasgos característicos de la deficiencia.

4.2.3. Clasificación de imágenes.

Finalizada la recolección y depuración de imágenes se obtuvo 6,090 imágenes agrupados en siete clases: Asintomáticas, Calcio, Potasio, Magnesio, Nitrógeno, Fósforo y Azufre. Se tiene 870 imágenes por clase y cada imagen contiene una única deficiencia, se puede encontrar la hoja sola o en el árbol. En el 95 % de imágenes se tiene la hoja completa, en el restante se puede observar hasta el 75 % del cuerpo de la hoja. que es básico para la clasificación de las hojas. En la figura 4.3 se muestra algunos ejemplos de las imágenes de *Persea Americana Hass Dataset*.



(a) Asintomática (b) Calcio (c) Potasio (d) Magnesio

(e) Nitrógeno (f) Fósforo (g) Azufre

Figura 4.3 Persea Americana Hass Dataset
Fuente: Elaboración propia.

4.2.4. Elaboración de dataset.

Concluida la división de imágenes por clase de *Persea Americana Hass Dataset* se realizaron los siguientes pasos:

1. Se segmento las 6,090 imágenes en dos grupos denominados *train (entrenamiento)* y *valid (validación)*, distribuidos con los porcentaje de 77 % y 23 % respectivamente [2]. En la tabla 4.3 se observa la distribución de la imágenes por clase.

Tabla 4.3 *Número de imágenes por clase.*

Clase	Denominación	Símbolo	Train	Valid	Total
0	Asintomática	A	670	200	870
1	Calcio	Ca	670	200	870
2	Potasio	K	670	200	870
3	Magnesio	Mg	670	200	870
4	Nitrógeno	N	670	200	870
5	Fósforo	P	670	200	870
6	Azufre	S	670	200	870
	Total		4,690	1,400	**6,090**

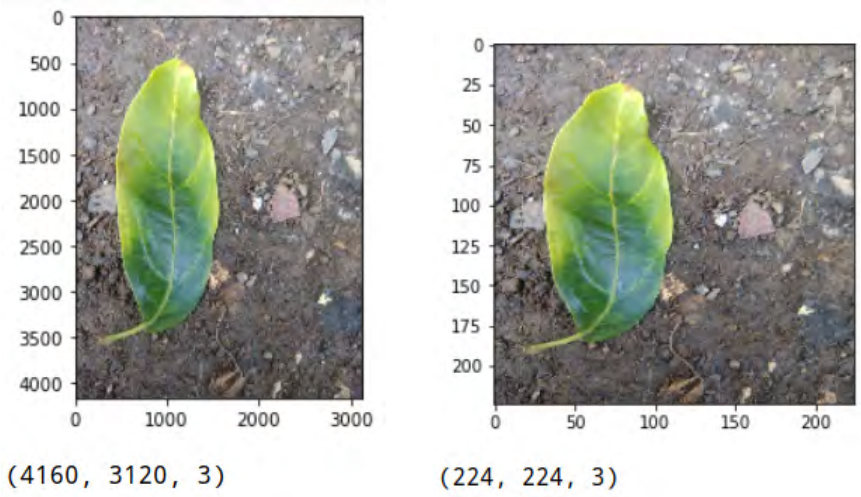
Fuente: Elaboración propia.

2. El *dataset* en su totalidad se alojó en un repositorio público de GitLab, el servicio ofrece 12Gb de almacenamiento y se realizó este proceso para tener disponibilidad del *dataset* en máquinas virtuales de *Google Collaboratory*.

3. Las imágenes que forman parte del *dataset* pasaron por un pre-procesamiento para convertir la imagen en un arreglo. A continuación se detalla el proceso de pre-procesamiento:

    - Primero se redimensionaron las imágenes, las arquitecturas CNN recibieron como entrada imágenes de 224×224 píxeles, a excepción de la arquitectura *Inception* que recibió como entrada imágenes de dimensiones de 229×229 pixeles.

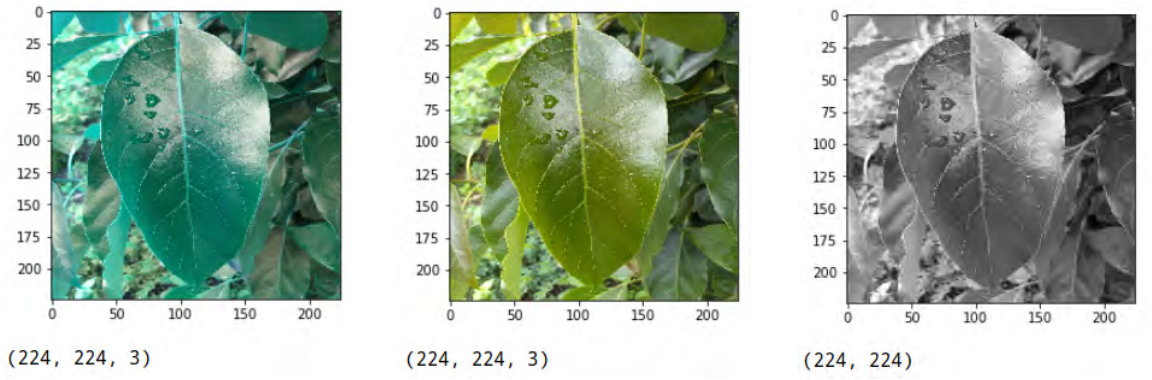
[2]La división del dataset es similar a la empleada en el trabajo de investigación: Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam et al. 2019. «Vietnamese Herbal Plant Recognition Using Deep Convolutional Features». International Journal of Machine Learning and Computing 9(3): 363-67.



(a) Imagen inicial (b) Imagen 224 x 224 pixeles

Figura 4.4 Redimensionar una imagen
Fuente: Elaboración propia.

- De las diversas escalas de colores (BGR [3], RGB, escala de grises, entre otros), que pueden ser aplicadas en las imágenes, se usó la escala de RGB porque en ella se puede apreciar los colores distintivos (características) presentes a causa deficiencias de macronutriente. Usar la escala de grises no hubiera sido apropiado para el desarrollo del proyecto debido a que en ella no se puede distinguir las tonalidades de los colores.

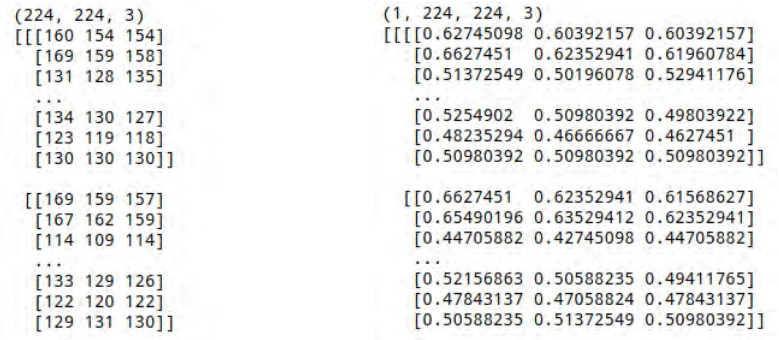


(a) BGR (b) RGB (c) Escala de grises

Figura 4.5 Escala de colores
Fuente: Elaboración propia.

[3]Escala utilizada por defecto de la librería OpenCV.

- El *shape* de los arrays son de $224 \times 224 \times 3$, el tres se refiere a los tres canales de colores RGB de la imagen con valores entre 0 y 255 como se muestra en la figura 4.6(a). Se normalizaron los valores de los píxeles para obtener valores entre 0 y 1, para ello se dividió el arreglo entre 255, el array resultante fue como en figura 4.6(b). Este proceso se denomina normalización de una imagen.



(a) Array inicial (b) Normalización

Figura 4.6 Normalización de imagen
Fuente: Elaboración propia.

4. Finalmente la imagen se convirtió en un arreglo de cuatro dimensiones: el primer elemento simboliza el tamaño del lote, el segundo y tercero representa el ancho y la altura respectivamente, la cuarta variable es la profundidad o número de canales.

5. La librería de Keras *ImageDataGenerator* simplificó el pre-procesamiento de las imágenes por lotes, realizándose internamente todo los pasos descritos en el numeral tres.

```
ImageDataGenerator( target_size=(224, 224),
                    color_mode='rgb',
                    rescale = 1./255)
```

4.2.5. Incremento del *Dataset*.

El conjunto de imágenes de *Persea Americana Hass Dataset* es de **6,090**, fue un número limitado para realizar un entrenamiento adecuado, contar con una gran cantidad de imágenes es crucial para un buen desempeño, por esta razón se utilizó generadores de imágenes que usan operaciones geométricas para generar nuevas imágenes. Las operaciones que se realizaron son:

- Traslación (vertical y horizontal).

- Rotación.

Se empleó la clase *ImageDataGenerator* de *keras* y se implementaron estas operaciones geométricas, los parámetros utilizados en el proyecto son:

```
ImageDataGenerator(rotation_range=20,
                   width_shift_range=0.2,
                   height_shift_range=0.2)
```

- ***rotation range:*** Los grados para la rotación aleatoria.

- ***width shift range:*** Fracción de ancho total que indica el rango de desplazamiento horizontales aleatorios.

- ***height shift range:*** Fracción de la altura total que indica el rango de desplazamientos verticales aleatorios.

4.3. Modelo de predicción

La problemática planteada en la investigación corresponde a la **clasificación de imágenes** [4], para la solución se utilizó las redes neuronales convolucionales profundas por su alto rendimiento en las tareas de visión artificial. El alto costo computacional del entrenamiento de un modelo CNN desde cero y el tamaño de *Persea Americana Hass Dataset*; fueron las principales razones para seleccionar la técnica ***Transfer Learning*** usando **modelos pre-entrenados.**

Se describe la secuencia de actividades realizadas: primero se seleccionó las arquitecturas de redes neuronales convolucionales pre-entrenadas, seguido se adoptó la estrategia de ajustes *(fine-tune)* para reentrenamiento y finalmente se fijaron parámetros de entrenamiento; los resultados obtenidos de esta etapa son los tres modelos de predicción (Inception-v3, DenseNet169 y MobileNet) entrenados en *Persea Americana Hass Dataset* para la identificación de deficiencias en hojas de la Persea Americana, los resultados obtenidos están expuestos en el próximo capítulo.

[4]El objetivo de la clasificación de imágenes es clasificar una imagen específica de acuerdo con un conjunto de categorías posibles.

4.3.1. Selección de arquitecturas de redes neuronales convolucionales.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [5], se desarrollo sobre un subconjunto de 1,000,000 imágenes distribuidas en 1,000 categorías del dataset *ImageNet*[6]. Desde el año 2010 se ha mejorado significativamente la exactitud y se redujo drásticamente la tasa de error; los grandes avances alcanzados fueron logrados aplicando redes neuronales convolucionales profundas.

Las *Convolutional Neural Network* (Redes Neuronales Convolucionales) se han convertido en la corriente principal en las tareas del reconocimiento de imágenes, desde que AlexNet popularizó las CNN al ganar el desafío ILSVRC 2012. Desde el 2014, la tendencia ha sido crear redes más profundas para lograr una mayor exactitud.

En el trabajo de investigación se seleccionaron tres arquitecturas con menor tasa de error y mayor rendimiento[7], con características que favorecieron en la solución del problema de investigación planteada. En las siguientes subsecciones se detallan las características de cada una de las arquitecturas seleccionadas con la finalidad de determinar adecuadamente los parámetros de entrenamiento.

4.3.1.1. *Inception-V3*

La arquitectura convolucional profunda *Inception* se presentó en ILSVRC 2014 como ***GoogLeNet***, también conocida como ***Inception-v1***[8], alcanzó una tasa de error top-5 de 6.67 % con convoluciones más pequeñas reduciendo la cantidad de parámetros a cuatro millones. Posteriormente realizaron modificaciones a la arquitectura Inception, en segunda iteración se incluye la **normalización de lotes** ***(Inception-v2*** [9]***)*** y la tercera iteración (***Inception-v3***) introdujo conceptos de **factorización de convoluciones**.

[5]Es el desafío anual de visión por computadora, el objetivo del concurso es desarrollar mejores técnicas para las tareas de clasificación, localización y detección de imágenes

[6]ImageNet está conformado por 14,197,122 imágenes, indexadas en 21,841 categorías (synsec).

[7]La comparativa de las arquitecturas se realiza en la tabla 4.13

[8]Características de Inception-v1 en *"Going Deeper with Convolutions" (Szegedy et al. 2015)*

[9]Características de Inception-v2 en *"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" (Ioffe and Szegedy 2015)*

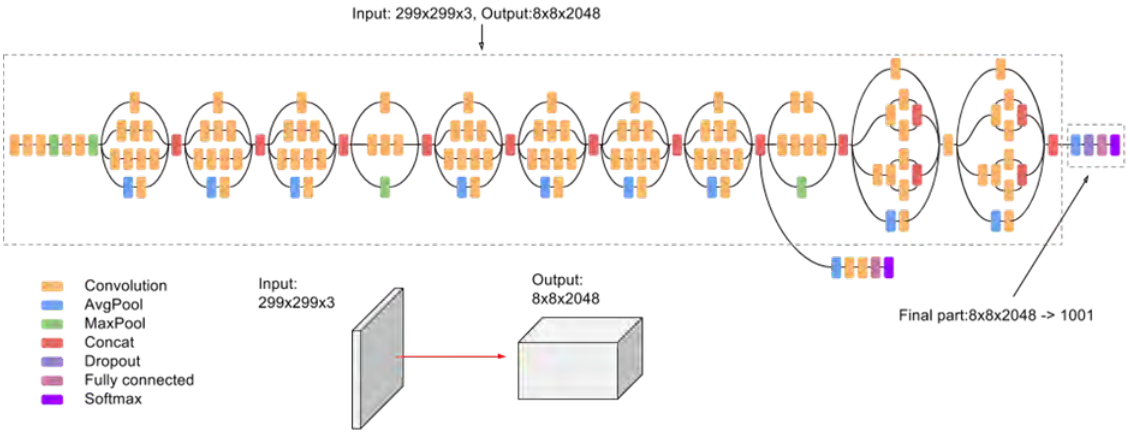
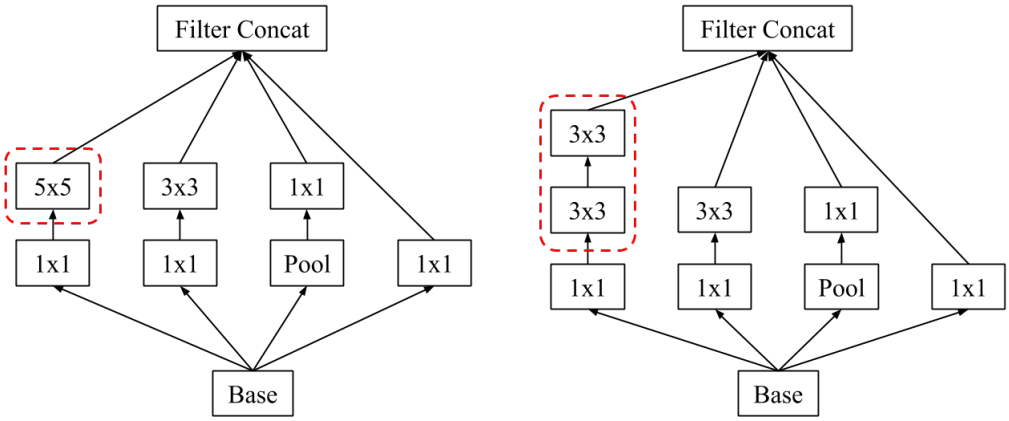


Figura 4.7 Diagrama de alto nivel del modelo Inception-v3.
Fuente: https://bit.ly/2S9FQEl

Inception redujo nueve veces la cantidad de parámetros con respecto a las arquitecturas predecesoras como AlexNet, que utilizó sesenta millones de parámetros y VGGNet empleo aproximadamente tres veces más parámetros que AlexNet; por las características el costo computacional de *Inception* es mucho más bajo. Por lo tanto, su uso es posible en escenarios con grandes volúmenes de datos o escenarios en los que la memoria o capacidad computacional son limitadas [Szegedy et al., 2016]. A continuación se describen las características resaltantes del modelo *Inception-v3*.

Factorización de convoluciones

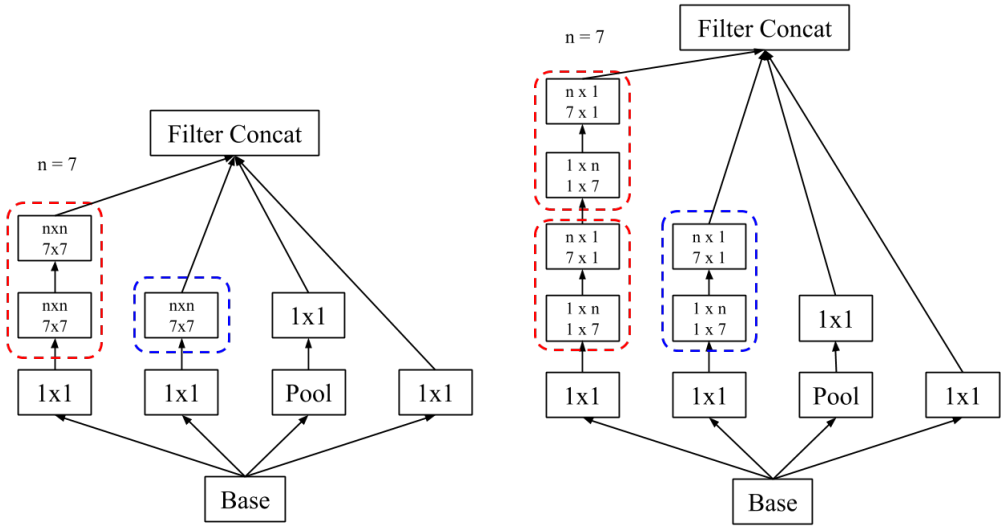
Las mejoras originales de GoogLeNet provienen de la reducción de dimensiones, empleando la factorización de convoluciones con la finalidad de **reducir el número de conexiones/parámetros** sin disminuir la eficiencia de la red. Las arquitecturas *Inception* son totalmente convolucionales, cada peso corresponde a una multiplicación por activación, por tanto las reducciones en el costo computacional resulta en un número reducido de parámetros.



(a) Módulo original de Inception-v1.

(b) Módulos de Inception-v3 donde cada convolución 5×5 se reemplaza por dos convolución 3×3.

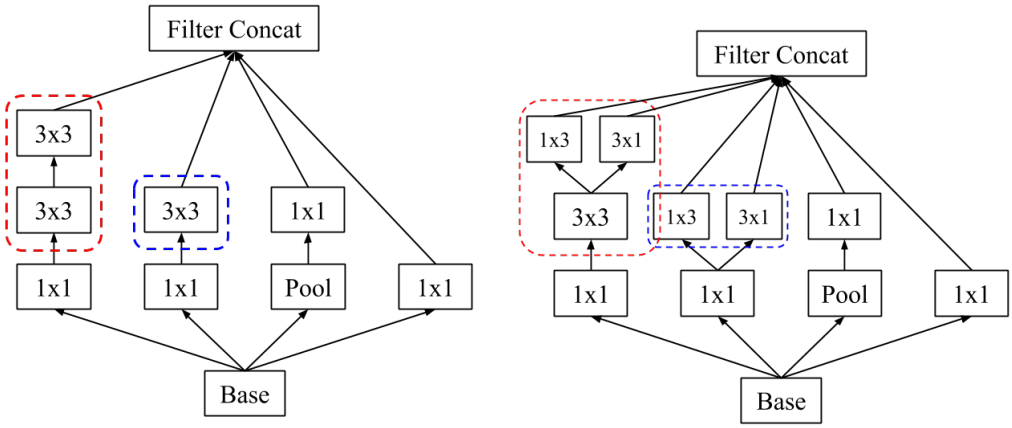
Figura 4.8 Factorización en convoluciones más pequeñas.
Fuente: [Szegedy et al., 2016].



(a) Factorización de $n \times n$ convoluciones, se reemplaza por una convolución $1 \times n$ seguida de una convolución $n \times 1$.

(b) Módulos de Inception-v3 después de la factorización.

Figura 4.9 Factorización en convoluciones asimétricas I.
Fuente: [Szegedy et al., 2016].



(a) Factorización para promover representaciones de alta dimensión.

(b) Módulos de Inception-v3 después de la factorización.

Figura 4.10 Factorización en convoluciones asimétricas II.
Fuente: [Szegedy et al., 2016].

Utilidad de los clasificadores auxiliares

En la arquitectura *Inception-v3* se utiliza un solo clasificador auxiliar a diferencia de la primera versión que empleaba dos clasificadores auxiliares. El clasificador auxiliar actúa como **regularizador**.

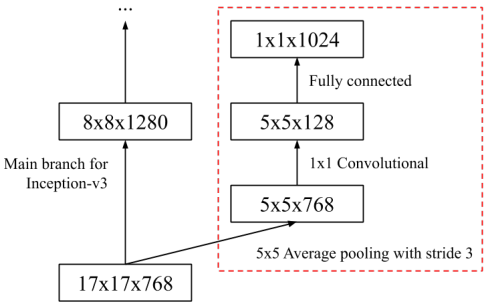


Figura 4.11 Clasificador auxiliar.
Fuente: "Rethinking the Inception Architecture for Computer Vision" [Szegedy et al., 2016].

Reducción eficiente del tamaño de grid

Tradicionalmente las CNN utilizan operaciones de agrupación máxima *(max-pooling)* el inconveniente es el costo de estas operaciones, *Inception-v3* propone la reducción del *grid* mientras expande los filtros con la reducción se logra una red menos costosa y más eficiente.

Arquitectura Inception-v3

El diseño de la red se muestra en la tabla 4.4, el tamaño de salida de cada módulo es el tamaño de entrada del siguiente. La red tiene una profundidad de 42 capas, el costo de cómputo es sólo alrededor de 2.5 más alto que el de GoogLeNet(Inception-v1) y aún es mucho más eficiente que VGGNet.

Tabla 4.4 *Arquitectura Inception-v3.*

type	patch size/stride or remarks	input size
conv	$3 \times 3/2$	$299 \times 299 \times 3$
conv	$3 \times 3/1$	$149 \times 149 \times 32$
conv padded	$3 \times 3/1$	$147 \times 147 \times 32$
pool	$3 \times 3/2$	$147 \times 147 \times 64$
conv	$3 \times 3/1$	$73 \times 73 \times 64$
conv	$3 \times 3/2$	$71 \times 71 \times 80$
conv	$3 \times 3/1$	$35 \times 35 \times 192$
3xInception	As in figure 4.8(b)	$35 \times 35 \times 288$
5xInception	As in figure 4.9(b)	$17 \times 17 \times 768$
2xInception	As in figure 4.10(b)	$8 \times 8 \times 1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

Fuente: "Rethinking the Inception Architecture for Computer Vision" [Szegedy et al., 2016].

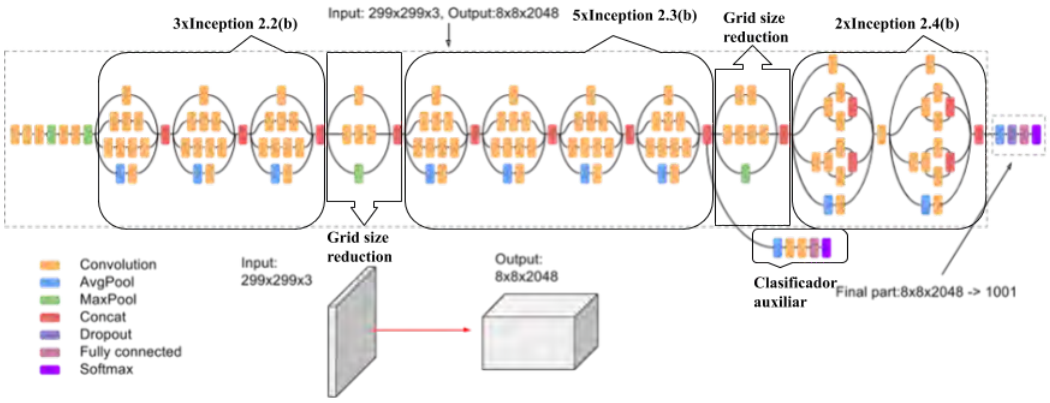


Figura 4.12 Diagrama de la arquitectura Inception-v3.
Fuente: Elaboración propia.

Evaluación y comparación con otros enfoques

Las evaluaciones se realizaron en los 48,238 ejemplos no incluidos en la lista negra en el conjunto de validación ILSVRC-2012. *Inception-v3* con 144 *crops* y cuatro modelos evaluados, se obtuvo una tasa de error top-5 de 3.58 % (Tabla 4.5). Las comparaciones se realizaron con resultados publicados en ILSVRC 2012.

Tabla 4.5 *Comparación de Inception-v3 con otro modelos.*

Network	**Models Evaluated**	**Crops Evaluated**	**Top-1 Error**	**Top-5 Error**
VGGNet	2	-	23.7 %	6.8 %
GoogLeNet (Inception-v1)	7	144	-	6.67 %
PReLU	-	-	-	4.94 %
BN-Inception (Inception-v2)	6	144	20.1 %	4.9 %
Inception-v3	4	144	**17.2 %**	**3.58 %**

Fuente: [Szegedy et al., 2016].

4.3.1.2. *DenseNet*

La arquitectura *DenseNet* ganó el premio *CVPR 2017 Best Paper Awards* desarrollado conjuntamente por la Universidad de Cornwell, la Universidad de Tsinghua y Facebook AI Research. *Dense Convolutional Network*, emplea una conexión densa para lograr menos parámetros y un alto rendimiento en comparación con *ResNet* y *Pre-Activation ResNet*. [Huang et al., 2017]

Dense Block

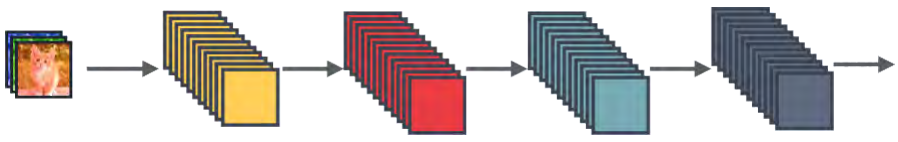


Figura 4.13 Red convolucional estándar.
Fuente: https://bit.ly/2NLssrE

En las redes neuronales convolucionales estándar, la imagen de entrada pasa por múltiples capas de convolución, con la finalidad de obtener características de alto nivel. A medida

que las CNN se hacen más profundas la gradiente tiende a disminuir o desaparecer al llegar al final de la red, topologías de red como *ResNet* [10] abordan este problema.

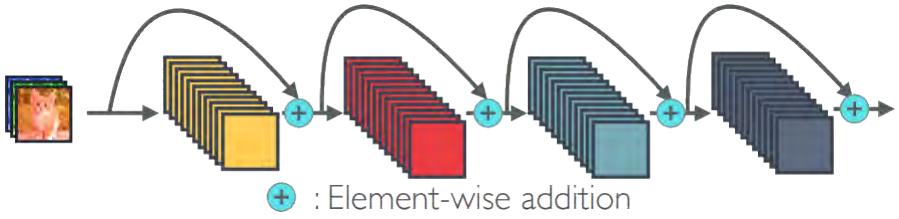


Figura 4.14 Concepto ResNet.
Fuente: https://bit.ly/2NLssrE

En ***ResNet***, propone el mapeo de identidad para acrecentar la gradiente de propagación, usando la adición de capas combina características. Usa algoritmos para pasar de un bloque de ResNet a otro [He et al., 2016].

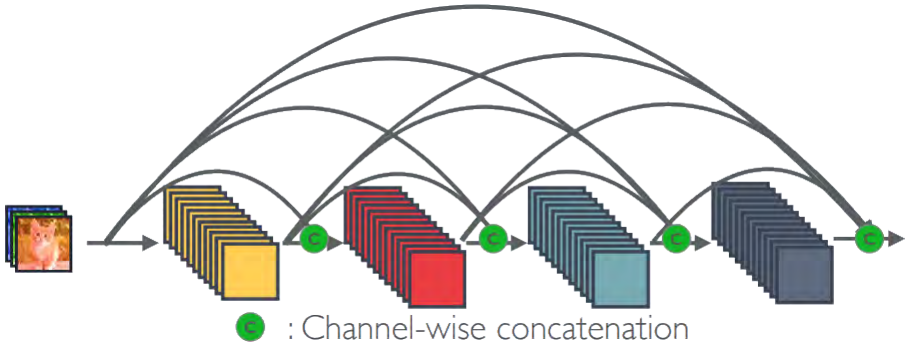


Figura 4.15 Dense Block.
Fuente: https://bit.ly/2NLssrE

En *DenseNet* para asegurar el máximo flujo de la información se conectan todas las capas directamente entre sí, la arquitectura recibe su nombre debido al patrón de conectividad. Cada capa obtiene entradas adicionales de todas las capas anteriores y pasa sus propios mapas de características a las capas posteriores, para combinar características se utiliza la **concatenación**, este proceso de flujo de información se denomina como **"conocimiento colectivo"**.

La conectividad de la red hace que se requieran menos parámetros a diferencia de los enfoques tradicionales, debido a que no es necesario volver aprender mapas de características redundantes. ResNet también hace explícita la preservación de la información a través

[10] Mas información sobre ResNet en *"Deep Residual Learning for Image Recognition"*

de la transformaciones de identidad aditivas, pero el número de parámetros es sustancialmente mayor debido a que cada capa tiene sus propios pesos, a diferencia la propuesta de *DenseNet* de agregar pequeños mapas de características al conocimiento colectivo y mantener los mapas de características restantes sin cambio, hace la red más delgada y compacta. Finalmente el clasificador toma una decisión basada en todos los mapas de características de la red.

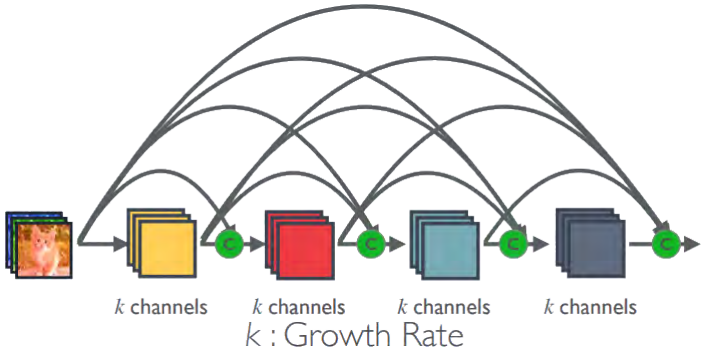


Figura 4.16 Arquitectura DenseNet.
Fuente: https://bit.ly/2NLssrE

Por tanto tiene mejor eficiencia de los parámetros, una gran ventaja de DenseNets es su flujo mejorado de información y gradientes en toda la red, lo que facilita su capacitación. Cada capa tiene acceso directo a los gradientes desde la función de pérdida y la señal de entrada original, lo que lleva a una supervisión profunda implícita. Esto ayuda a la formación de arquitecturas de redes más profundas. Además, también observamos que las conexiones densas tienen un efecto de regularización, lo que reduce el sobreajuste en tareas con tamaños de conjuntos de entrenamiento más pequeños. Además tiene mayor eficiencia computacional y de memoria [Huang et al., 2017].

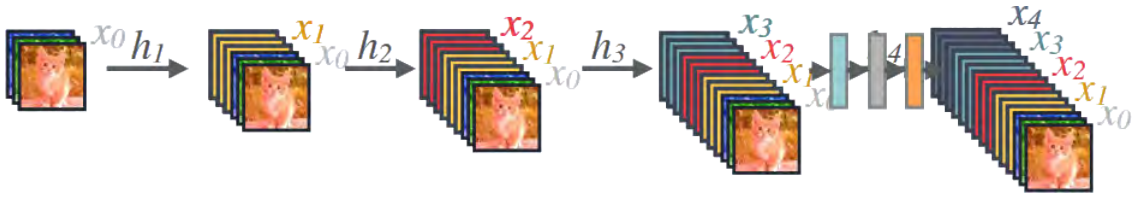


Figura 4.17 Concepto de concatenación "Conocimiento colectivo."
Fuente: https://bit.ly/2NLssrE

DenseNets tiene varias ventajas convincentes: alivian el problema del gradiente de fuga, refuerzan la propagación de características, fomentan la reutilización de características y reducen sustancialmente la cantidad de parámetros.

Arquitectura *DenseNet*.

1. **Conectividad densa.** Para mejorar más el flujo de información entre capas, *DenseNet* propone un patrón de conectividad diferente: introduce conexiones directas de cualquier capa a todas las capas posteriores. En consecuencia ℓ^{th} recibe características de todas las capas predecesoras $x_0, \ldots, x_{\ell-1}$, como *input*:

$$x_\ell = H_\ell([x_0, x_1, \ldots, x_{\ell-1}]), \tag{4.1}$$

   Donde $[x_0, x_1, \ldots, x_{\ell-1}]$ se refiere a la concatenación de los mapas de características producidos en las capas $0, \ldots, \ell-1$.

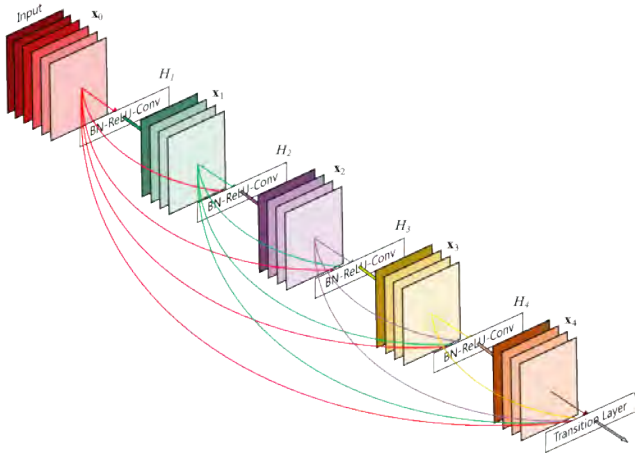


Figura 4.18 Esquema de la arquitectura DenseNet
Fuente: "Densely Connected Convolutional Networks" [Huang et al., 2017].

2. **Función compuesta.** Definimos H_ℓ como una función compuesta por tres operaciones consecutivas: Normalización de lotes (BN), seguida por rectificación lineal (ReLU) y una convolución de 3×3 (Cov).

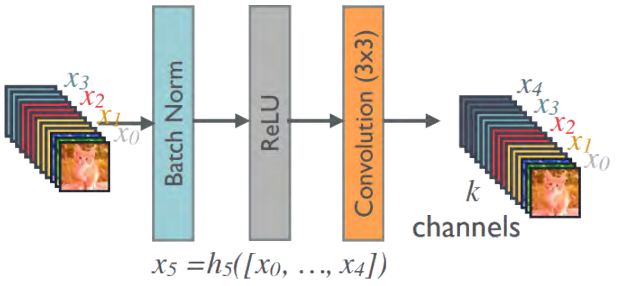


Figura 4.19 Composición de una capa en DenseNet
Fuente: https://bit.ly/2NLssrE

3. ***Pooling* o Agrupación de capas.** La operación de concatenación de la ecuación 4.1 no es viable cuando cambia el tamaño de los mapas de características. Para ello se divide la red en múltiples bloques densos densamente conectados. A las capas entre los bloques se les denomina capas de transición que se encargan de la convolución y agrupación. Las capas de transición utilizadas son una capa de normalización de lotes (BN) y una capa de convolución 1×1 seguida de una capa de *averange pooling* de 2×2.

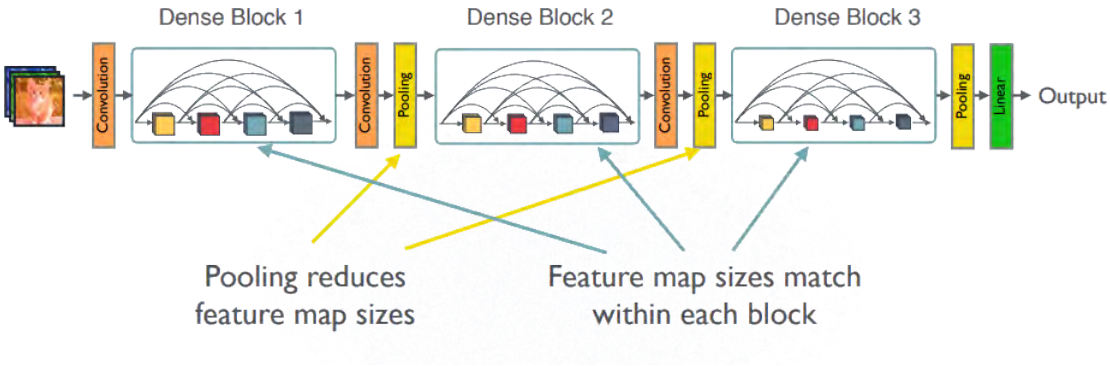


Figura 4.20 Múltiples Dense Blocks
Fuente: https://bit.ly/2NLssrE

4. **Tasa de crecimiento.** Si la función H_ℓ produce k mapas de características, se deduce que la capa ℓ^{th} tiene $k_0 + k \times (\ell - 1)$ mapas de características de entrada, donde k_0 es el número de canales en la capa de entrada. El hiperparámetro k es la tasa de crecimiento de la red, regula la cantidad de información nueva que cada capa contribuye al estado global.

5. **Cuellos de botella.** Aunque cada capa produce k mapas de características de salida, normalmente se tiene más entradas. Por lo observado en las arquitecturas *Inception* y *ResNet* que una capa de convolución 1×1 se puede agregar como una capa de cuello de botella antes de cada convolución de 3×3 para reducir de número de mapas de características de entrada y por tanto mejorar la eficiencia computacional. A la versión de la red se le denomina *DenseNet-B*, cada convolución 1×1 produce $4k$ mapas de características.

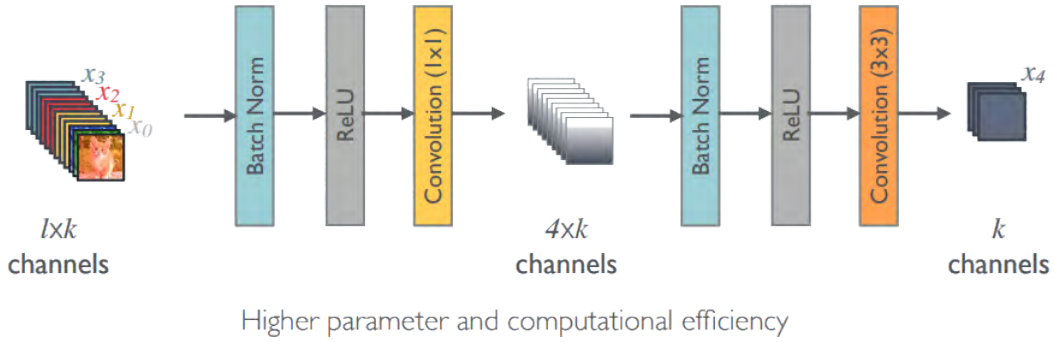


Figura 4.21 DenseNet-B
Fuente: https://bit.ly/2NLssrE

6. ***Compression* o Comprimir la red.** Para mejorar aún más la densidad del modelo, podemos reducir la cantidad de mapas de características en las capas de transición. Si un bloque denso contiene m mapas de características, permitimos que la siguiente capa de transición genere $[\theta\ m]$ mapas de salida, donde $0 < \theta \leq 1$ se denomina factor de compresión. Cuando $\theta = 1$, el número de mapas de características en las capas de transición no cambia. Se refiere a *DenseNet* con $\theta < 1$ como *DenseNet-C*, establecemos $\theta = 0{,}5$. Cuando se utilizan tanto el cuello de botella como las capas de transición con $\theta < 1$, se denomina al modelo como *DenseNet-BC*.

Comparación con otros enfoques.

Los resultados obtenidos del entrenamiento con ImageNet son de la estructura de *DenseNet-BC* con cuatro bloques densos con imágenes de entrada de 224×224 pixeles, con diferentes profundidades y tasas de crecimiento. Se comparó con la arquitectura de ResNet. Las tasas de errores de validación se muestran en la tabla 4.6.

Tabla 4.6 *Tasas de errores Top-1 y Top-5 en conjunto de validación de Imagenet*

Model	top-1	top-5
DenseNet-121	25.02 / 23.61	7.71 / 6.66
DenseNet-169	23.80 / 22.08	6.85 / 5.92
DenseNet-201	22.58 / 21.46	6.34 / 5.54
DenseNet-264	22.15 / 20.80	6.12 / 5.29

Fuente: "Densely Connected Convolutional Networks" [Huang et al., 2017].

La Figura 4.22 muestra las tasas de error top-1 de *DenseNets* y *ResNets* en el conjunto de

datos de validación de *ImageNet* en función del número de parámetros (izquierda) y FLOPs durante el tiempo de prueba (derecha).

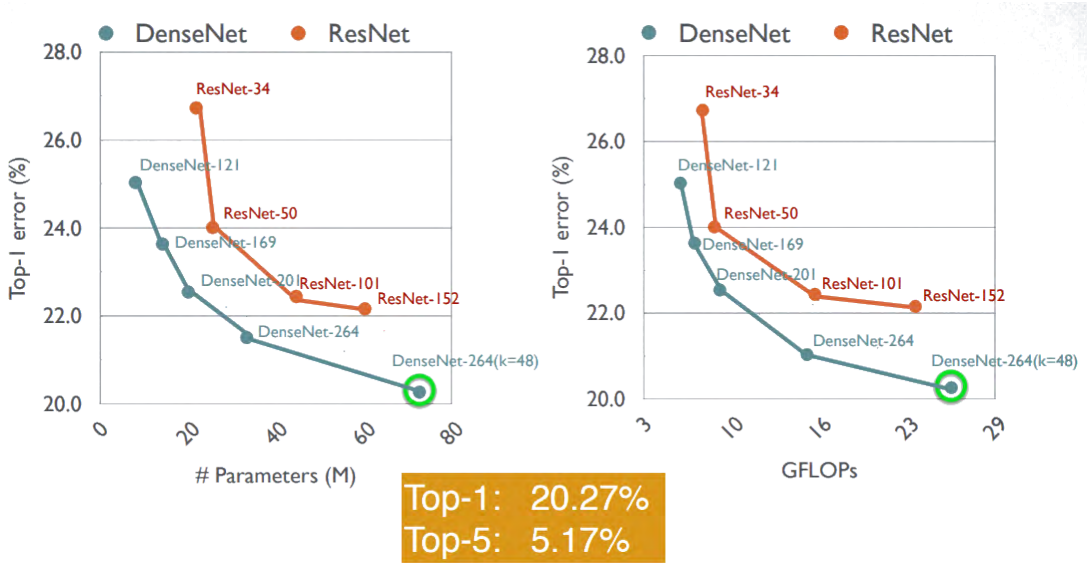


Figura 4.22 Resultados comparados con ResNet
Fuente: https://bit.ly/2NLssrE

Los resultados representados en la figura revelan que *DenseNet* se desempeña a la par con los *ResNets* de última generación, mientras que requiere significativamente menos parámetros y cálculos para lograr un rendimiento comparable.Por ejemplo, un modelo DenseNet-201 con parámetros de 20M produce un error de validación similar al de una ResNet de 101 capas con más de 40M de parámetros. Tendencias similares desde el panel derecho, que traza el error de validación en función del número de FLOP: una DenseNet que requiere tanto cómputo como una ResNet-50 que funciona a la par con una ResNet-101, que requiere el doble cálculo.

De los experimentos realizados por los autores DenseNet-264 $(k = 48)$ obtuvo mejores resultados: 20.27 % de error Top-1 y 5.17 % de error Top-5.

4.3.1.3. *MobileNet*

También conocida como *MobileNetv1* fue propuesta por Google en el año 2017, se construyen principalmente a partir de **convoluciones separables en profundidad** *(Depthwise Separable Convolution)* para disminuir significativamente el número de parámetros y la complejidad del modelo, la arquitectura reduce la latencia. Adicionalmente se emplean dos hiper-parámetros: **Multiplicador de ancho** ***(Width multiplier)*** α y **Multiplicador de resolución** ***(Resolution Multiplier)*** ρ. MobileNet se adecua a requisitos de diseño para aplicaciones de visión en dispositivos móviles e integradas. [Howard et al., 2017]. Posteriormente se describe las capas centrales *Depthwise Separable* Convolution y los hiper-parámetros.

Depthwise Separable Convolution

MobileNet utiliza una forma particular de factorizar convoluciones en dos capas, la factorización de una convolución estándar está dado por una capa **convolución en profundidad** *(Depthwise convolutions)* seguida de una convolución de 1×1 denominada **convolución puntual** *(Pointwise convolutions)*.

1. **Convolución estándar** es una capa de convolución estándar tiene como input $D_F \times D_F \times M$ un mapa de características **F** que genera $D_G \times D_G \times N$ un mapa de características **G**; donde D_F es el tamaño del mapa de características de entrada, M es el número de canales de entrada *(input depth)*, D_G es el tamaño del mapa de características de salida y N es el número de canales de salida (*output depth*). La capa de convolución estándar está parametrizada por el kernel de convolución **K** de dimensiones $D_K \times D_K \times M \times N$, donde D_K es el tamaño del kernel. El costo computacional de convolución estándar es:

   $$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{4.2}$$

   En la operación de convolución estándar se realiza el filtrado y combinación de características para generar una nueva representación, la operación se puede dividir en dos pasos utilizando la factorización de convolución denominada *Depthwise Separable Convolution* con el objetivo de reducir el costo computacional.

2. **Convolución en profundidad** ***(Depthwise convolutions)*** es más eficiente que la convolución estándar. Se usa la convolución por profundidad para aplicar un solo filtro por cada canal de entrada (*input depth*). Sin embargo, solo filtra los canales de entrada, pero no los combina para crear nuevas características es por ello que requiere de

una capa adicional. El costo computacional de convolución en profundidad es:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \tag{4.3}$$

3. **Convolución puntual** ***(Pointwise convolutions)*** es una simple convolución de 1×1, se usa para crear una combinación lineal para la salida de la convolución de profundidad. Es necesaria para generar nuevas características.

La combinación de la convolución en profundidad y convolución de 1×1 se denomina convolución separable por profundidad. Costo computacional de la convolución separable por profundidad:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \tag{4.4}$$

Es igual a la suma del costo de la convolución por profundidad y convolución puntual. La reducción de cálculo es:

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2} \tag{4.5}$$

MobileNet utiliza convoluciones separables por profundidad de 3×3, que usa entre ocho y nueve veces menos cálculo que las convoluciones estándar con una pequeña reducción en la precisión.

Arquitectura *MobileNet*

Todas las capas de convolución van seguidas de la normalización de lotes, *batchnorm* (BN), y ReLU, excepto la capa final totalmente conectada que se alimenta de la función *softmax* para la clasificación. MobileNet tiene 28 capas, para el entrenamiento se utilizaron menos técnicas de regularización y aumento de datos por que los modelos pequeños tienen menos problemas con el sobreajuste.

Tabla 4.7 *Arquitectura MobileNet.*

Type / Stride	**Filter Shape**	**Input Size**
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Fuente: [Howard et al., 2017]

Tabla 4.8 *Depthwise Separable vs Fully Convolution MobileNet*

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7 %	4866	29.3
MobileNet	70.6 %	569	4.2

Fuente: [Howard et al., 2017]

El uso de convoluciones separables en profundidad en comparación con las convoluciones completas (Tabla 4.8) se tuvo una pérdida en la precisión en un 1 % en ImageNet, pero

los agregados múltiples y los parámetros se reducen enormemente.

Multiplicador de ancho *(Width multiplier)* α : Modelos más delgados

Para construir modelos más pequeños y menos costosos computacionalmente, se utilizó el parámetro α denominado multiplicador de ancho, el rol es adelgazar uniformemente cada capa. Dado un multiplicador de ancho α, el número de canales de entrada M se convierte en αM y el número de canales de salida N se convierte αN.

El costo computacional de una convolución separable en profundidad con multiplicador de ancho α es:

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F \tag{4.6}$$

donde, $\alpha \in (0,1]$ con ajustes de 1, 0.75, 0.5 y 0.25. Con $\alpha = 1$ es la línea base de MobileNet y $\alpha < 1$ se reducen las *MobileNets*. El costo computacional y el número de parámetros se pueden reducir de forma cuadrática aproximadamente α^2.

Tabla 4.9 *MobileNet Width Multiplier*

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Fuente: [Howard et al., 2017]

Multiplicador de resolución *(Resolution Multiplier)* ρ: representación reducida

Se aplica para controlar la resolución de la imagen de entrada y la representación interna en cada capa reducida por el mismo multiplicador, en la práctica se puede configurar implícitamente ρ configurando la resolución de entrada.

Se puede expresar el costo computacional para las capas centrales de la red como convoluciones separables en profundidad con multiplicador de ancho α y el multiplicador de resolución ρ:

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F \tag{4.7}$$

donde, $\rho \in (0,1]$ y la resolución de entrada de la red es 224, 192, 160 ó 128. Cuando $\rho = 1$ es la linea de la *MobileNet* y si $\rho < 1$ se reduce el cálculo de *MobileNets*. El multiplicador de resolución reduce el costo computacional en ρ^2.

Tabla 4.10 *MobileNet Resolution Multiplier*

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

Fuente: [Howard et al., 2017]

Comparación con otros enfoques

La tabla 4.11 compara la version completa de *MobileNet* **(1.0 MobileNet-224)** con GoogLeNet (Ganador de ILSVRC 2014) y VGG16(1º finalista de ILSVRC 2014). *MobileNet* es casi tan preciso como VGG16, 32 veces más pequeño y menos tiempo de computo. Es más preciso que GooLeNet al mismo tiempo que es más pequeño y con 2.5 veces menos de cálculos.

Tabla 4.11 *Comparación de MobileNet con otros modelos*

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Fuente: [Howard et al., 2017]

La tabla 4.12 compara un versión reducida de *MobileNet* **(0.50 MobileNet-160)**, con multiplicador de ancho $\alpha = 0{,}5$ y resolución de 160×160. La versión reducida es 4 % mejor que AlexNet, mientras que es 45 veces mas pequeña y 9.4 veces menos cálculo; también es un 4 % mejor que *Squeezenet* con aproximadamente el mismo tamaño y 22 veces menos cálculos.

Tabla 4.12 *Comparación Smaller MobileNet con otros modelos*

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Fuente: [Howard et al., 2017]

4.3.2. Comparación de los modelos pre-entrenados.

Se elaboró una tabla comparativa en base a las características de las arquitecturas seleccionadas.

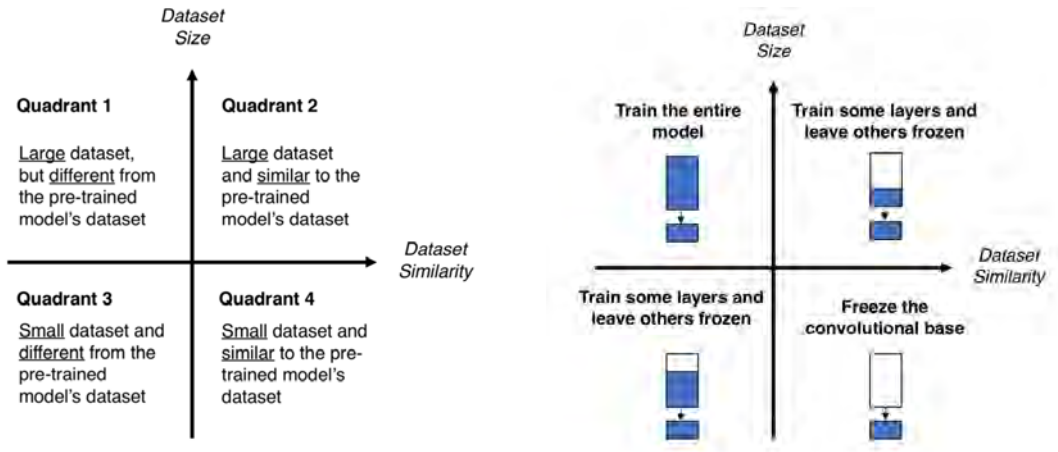
Tabla 4.13 *Comparativa de los modelos Inception-v3, DenseNet169 y MobileNet*

Modelo	Año	Profundidad	# Parametros	Top-5 (%) error	Input size
Inception-v3	2015	42	24M	4.2 %	$229 \times 229 \times 3$
DenseNet169	2017	100	1.3M	5.9 %	$224 \times 224 \times 3$
MobileNet	2017	28	4.2M	10.1 %	$224 \times 224 \times 3$

Fuente: Elaboración propia.

4.3.3. Estrategia de ajuste para los modelos pre-entrenados.

La estrategia adoptada para el problema de investigación se hizo en base al tamaño del conjunto de datos y la similitud con el conjunto de datos en los que se entrenó los modelos pre-entrenados utilizados; estos dos aspectos se representan en una Matriz Tamaño-Similitud (Figura 4.23(a)), que clasifica en cuatro cuadrantes los problemas de clasificación de imágenes y propone la estrategia a emplearse como se muestra en la figura 4.23(b).



(a) Matriz Tamaño-Similitud (b) Mapa de decisión para ajustes

Figura 4.23 Clasificación del problema de investigación.
Fuente: https://bit.ly/2yWyoDq

Persea Americana Hass Dataset compuesto por 6,090 imágenes distribuidas en 870 imágenes por cada una de las siete clases; es considerado como un ***dataset pequeño***. Los

modelos pre-entrenados seleccionados fueron entrenados previamente con el *dataset* ImageNet en subconjunto de 1,000 clases, sin embargo el subconjunto de categorías no incluye a hojas de la Persea Americana; en consecuencia **no existe similitud** entre *Persea Americana Hass Dataset y el dataset ImageNet*. Por las características del problema planteado la estrategia empleada está ubicada en el tercer cuadrante de la Matriz Tamaño-Similitud.

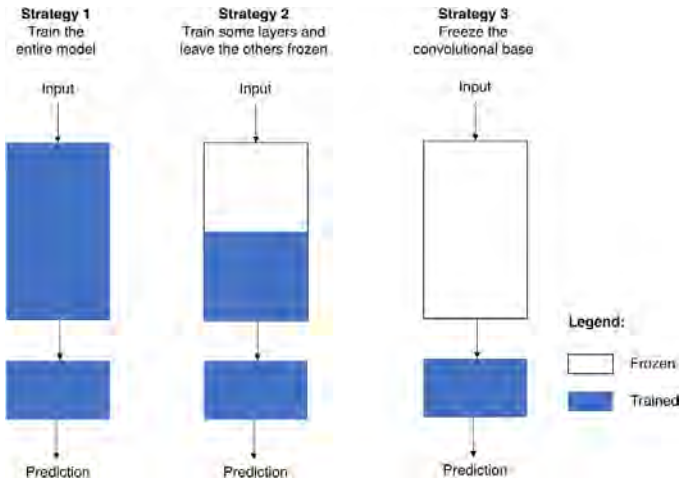


Figura 4.24 Estrategias para ajuste de reentrenamiento.
Fuente: https://bit.ly/2yWyoDq

Una arquitectura simplificada de un modelo convencional basado en CNN tiene dos partes: **base convolucional** y el **clasificador**, se adoptó la segunda estrategia *(Figura 4.24)*; entrenar algunas capas y congelar otras. Adicionalmente se reemplazó el clasificador de la última capa por una capa *softmax* de siete clases. Para encontrar el modelo con mejor rendimiento se probó distintas combinaciones de capas congeladas y entrenadas. Si se profundiza demasiado el modelo tendrá un problema de **sobreajuste**, en contraparte si se entrena superficialmente los modelos no aprenderán nada útil.

4.3.4. Determinar parámetros de entrenamiento.

Se ajustaron los parámetros en base a las características de los modelos seleccionados con la finalidad de obtener un mejor rendimiento. Los parámetros con los que se realizaron las pruebas son:

- **Número de *epochs*:** Número de veces que se pasó como *input* el conjunto de entrenamiento a la red neuronal. Este parámetro se define en 50 épocas para *Inception y MobileNet*, mientras que para *DenseNet* en 20 épocas.

- **Tamaño de la imagen:** Las imágenes del *dataset* fueron redimensionadas al tamaño de 224×224 pixeles.
- **Número de canales:** Las imágenes del *dataset* están en la escala de colores **RGB** debido a que las entradas de los modelos deben tener tres canales ($224 \times 224 \times 3$)
- **Número de neuronas de la capa intermedia *fully-connected*:** La capa previa a la capa de salida tuvo 1024 neuronas.
- **Función de activación para la capa de salida:** La función aplicada en el clasificador es *softmax* de siete clases.
- **Función de optimización:** La función de optimización usada es *Adam*. Adam recibe como entrada: *learning Rate*, que fue fijado a 0.001, $beta_1a$0,9, $beta_2a$0,999 y *decay*0,0. Estos parámetros vienen estimados por defecto.

Tabla 4.14 Parámetros de entrenamiento

Parámetros de Entrenamiento	**InceptionV3**	**DenseNet**	**MobileNet**
Número epochs	50	20	50
Tamaño imagen	229×229	224×224	224×224
Escala de Colores	RGB	RGB	RGB
Número de Canales	3	3	3
Número de neuronas de la capa intermedia *fully-connected*	1024	1024	1024
Función de activación para la capa de salida	softmax	softmax	softmax
Función de optimización	Adam	Adam	Adam

Fuente: Elaboración propia.

4.3.5. Implementación.

Con los modelos ya seleccionados, adoptado la estrategia para el entrenamiento y ajustados los parámetros en este apartado se presentan las plataformas empleadas en la implementación de las redes neuronales convolucionales y el proceso de implementación realizado.

4.3.5.1. Recursos empleados.

Tabla 4.15 *Recursos empleados en la construcción de las CNN.*

Lenguaje de programación	Python 3.6.5
Librerias	TensorFlow
	Keras
Entorno	Jupiter Notebook
	Google Colaboratory

Fuente: Elaboración propia.

La disponibilidad de usar los recursos de *Google Collaboratory* como GPU potente y 12 GB de memoria RAM, posibilitó probar una mayor número de modelos en búsqueda de un solución con mejor desempeño.

4.3.5.2. Implementación de los modelos.

Se importó la arquitectura pre-entrenado *(ModelPreTrained)* con los pesos del *dataset* ImageNet *(weigths = 'imagenet')*, excluyendo la última capa de 1,000 neuronas *(include_top = False)*.

En la linea 2, reemplazar ***ModelPreTrained*** por uno de los modelos pre-entrenados seleccionados ***InceptionV3, DenseNet169 o MobileNet***, se agregaron capas densas con la función de activación ReLU.

Finalmente se agregó el clasificador que es la última capa de la red, y debe tener tantas neuronas como el número de clases a identificar; para la investigación son siete clases (A, Ca, K, Mg, N, P y S) con la función *softmax* (Línea 9).

```
#create the base pre-treined model
base_model = ModelPreTrained(weigths = 'imagenet', include_top = False)
x = base_mode.output
x = GlobalAveragePooling2D()(x)
#add fully-connected layers
x = Dense(1024, activation='relu')(x)
x = Dense(512, activation='relu')(x)
#add classifier layer with softmax activation
preds = Dense(7, activate='softmax')(x)
```

Posterior a la construcción de un nuevo modelo con los requerimientos de la investigación, se creó el modelo en base a la arquitectura implementada en el fragmento de código anterior especificando las entradas y salidas.

```
model=Model(inputs=base_model.input,outputs=preds)
```

Se configuro los pesos de cada capa del modelo pre-entrenado para definir las capas que no se entrenan, es decir, se congelaron los pesos y sesgos para mantener la información que ya aprendió a través del entrenamiento inicial de *ImageNet* (Línea de código 2). Las capas de entrenamiento aprenden nuevas características del conjunto de datos *Persea Hass Dataset* (Línea de código 4).

Es importante el equilibrio entre ambas. Si la cantidad de capas congeladas es mayor el modelo no aprenderá nada útil, caso contrario se presenta un problema de sobreajuste. Se entrenaron tres combinaciones por cada modelo (Configuración entre capas se observa en tabla 4.16).

```
for layer in model.layers[:layers]:
    layer.trainable=False
for layer in model.layers[layers:]:
    layer.trainable=True
```

Recompilar el modelo; con función de activación *Adam*, la función Loss *categorical_crossentropy* y las métrica para juzgar el rendimiento de los modelos *accuary*.

```
#compile model
model.compile(optimizer = 'Adam',
              loss = 'categorical_crossentropy',
              metrics = ['accuary'])
```

4.3.6. Entrenar los modelos de predicción.

En esta fase se entrenó el modelo con *Persea Americana Hass Dataset*. En la línea 3 se definió la cantidad de épocas que se entrenó el modelo. Finalizado el entrenamiento se guardaron los modelos construidos (Linea 6), son usados en el prototipo. Los resultados se muestran en el capítulo posterior.

```
model.fit_generator(generator=train_generator,
                    steps_per_epoch=step_size_train,
                    epochs=30,
```

```
                validation_data=validacion_generador,
                validation_steps=5)
model.save('ModelName.model')
```

Tabla 4.16 *Configuración para entrenamiento.*

Arquitectura	**Modelo**	**Capas Congeladas**	**Épocas**
InceptionV3	*A*	101	50
	B	165	50
	C	229	50
DenseNet169	*A*	474	20
	B	368	20
	C	274	20
MobileNet	*A*	40	50
	B	50	50
	C	60	50

Fuente: Elaboración propia.

4.4. Prototipo

En esta sección se presenta la plataforma escogida para el desarrollo del prototipo, el cual cuenta con una arquitectura como se muestra en la figura 4.25.

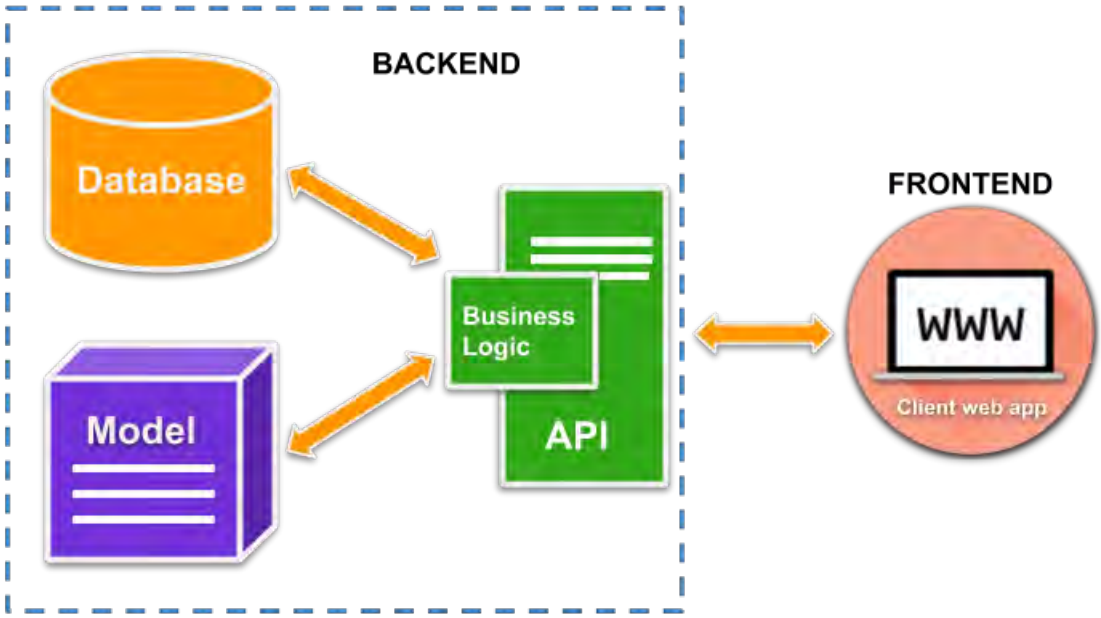


Figura 4.25 Arquitectura del prototipo
Fuente: Elaboración propia

La arquitectura utilizada para el desarrollo del prototipo se divide en dos sectores. El *Backend* para englobar los componentes del prototipo: la base de datos *(Database)*, modelo de predicción *(Model)* y parte lógica *(API)* que está construido en el micro framework *Flask* que proporciona la interacción de los componentes según lo requerido por el cliente web. El *Frontend* está compuesto por el cliente web para establecer comunicación a la parte lógica del *Backend* para realizar las predicciones.

4.4.1. Backend.

El *Backend* proporciona la parte lógica del sistema, a continuación se detalla la implementación.

4.4.1.1. Base de datos *(Database).*

La base de datos que utiliza el prototipo está implementado en el gestor datos no relacional *MongoDB*, para guardar los datos en estructuras de datos BSON (JSON Binario) con un esquema dinámico, logrando que la integración de datos en la aplicación sea rápida.

Para el prototipo se utiliza la base de datos ***db_persea***, este almacena los datos que son consultados por el prototipo según sea el tipo de deficiencia que corresponda a la clasificación realizada. El diagrama de la base de datos se muestra en la figura 4.26.

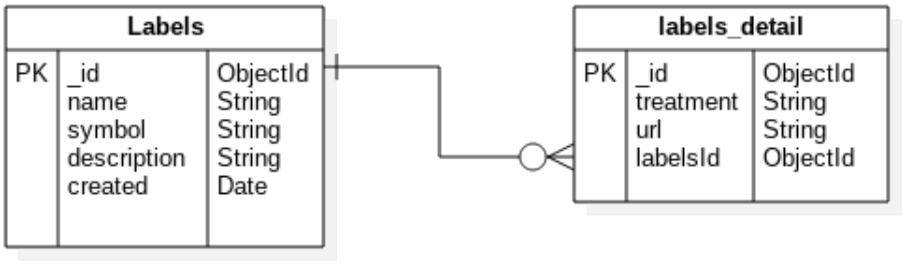


Figura 4.26 Diagrama de base de datos *db_persea*
Fuente: Elaboración propia

Se presentan dos entidades :

1. **labels (etiquetas)**. Almacena los datos de la deficiencia de macronutriente que afecta a la persa americana con los atributos de nombre, símbolo y descripción.

2. **labels_detail (etiquetas_detalle)**. Almacena los datos del tratamiento que puede ser empleado para tratar la deficiencia del macronutriente.

4.4.1.2. Modelo de clasificación (Model).

El prototipo utiliza tres modelos de clasificación, que permite detectar a que tipo de deficiencia de macronutriente corresponde, al proporcionar una imagen con características similares que presentan dichas deficiencias.

Los modelos utilizados para la clasificación son *Inception*, *DenseNet* y *MobilNet*. Estos modelos se encuentran ubicadas en el *Backend* y son utilizados en el procesos de clasificación.

4.4.1.3. API.

El *API* está desarrollado en el micro framework *Flask* escrito en *Python* con el que se implementó la aplicaciones web, empleando librerías y utilizando los modelos entrenados para la clasificación de macronutrientes.

Tabla 4.17 *Recursos empleados en desarrollo del backend*

Lenguaje de programación	Python 3.6.5
Microframework	Flask
	Flask-RESTful
	Flask-Cors
Librerias	TensorFlow
	Opencv-Python
	Numpy
Base de Datos	MongoDB
	Mongoengine

Fuente: Elaboración propia.

Estructura del API.

El *API* construido presenta una distribución de carpetas, para proporcionar un orden en desarrollo del *Backend*, esta distribución tiene archivos de conexión a la base de datos, *urls* de solicitud de procedimientos, modelos que representan tablas en la base de datos y controladores.

En la figura 4.27 se representa la estructura de directorios y archivos que conforman la aplicación. Con borde de color verde están los archivos de conexión a la base de datos *(.env)* y la figuración del servidor *(application.py)*, de color rojo archivos que se emplearon para el modelado de la base de datos, la declaración de la url que utiliza la API para procesar las solicitudes están agrupadas de un color morado y finalmente el color azul corresponde a los archivos donde se implementó la parte lógica del software es decir los controladores.

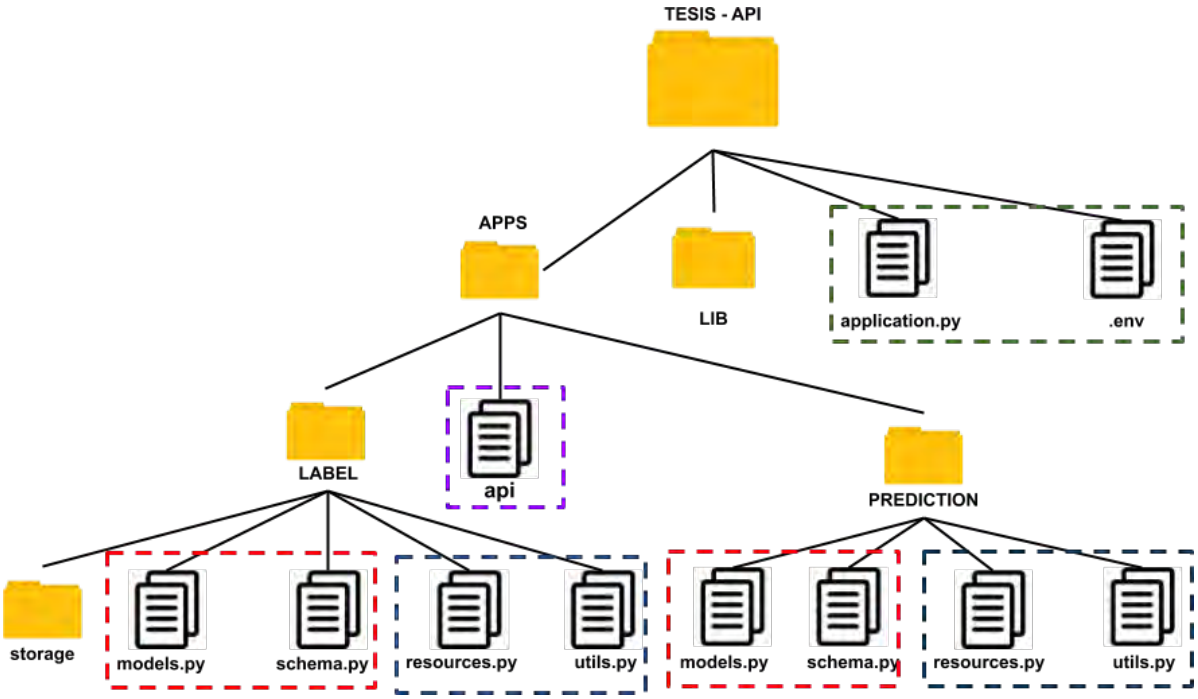


Figura 4.27 Estructura directorios y archivos del *API*
Fuente: Elaboración propia

Conexión a la base de datos y configuración del servidor.

Para configurar la conexión de base de datos y del servidor se utilizó una lista de variables de entorno en la que se declara la información de la aplicación como se muestra en la figura 4.28.

```
1  DEBUG=True
2  APP_PORT=5000
3  FLASK_APP="application.py"
4  FLASK_ENV="development"
5  SECRET_KEY="hard-secret-key"
6  MONGODB_URI="mongodb://localhost:27017/db_persea"
```

Figura 4.28 Variables de entorno *(.env)*
Fuente: Elaboración propia

Modelos de entidades.

Los archivos *models.py* proporcionan información de la base de datos, contiene los atributos, tipos y características de los datos almacenados, cada modelo le pertenece a una única colección, cada atributo del modelo representa un campo dentro de la base de datos.

```
16  class LabelMixin(db.Document):
17
18      """
19      implementacion por defecto para campos de label
20      """
21      meta = {
22          'abstract':True,
23          'ordering':['symbol']
24      }
25
26      symbol = StringField(required=True, max_length=5)
27      description = StringField(required=True)
28      created = DateTimeField(default=datetime.now)
29      active = BooleanField(default=False)
30
31      def is_activate(self):
32          return self.active
```

Figura 4.29 *models.py*
Fuente: Elaboración propia

La figura 4.29 se asignó los atributos de la clase *LabelMixin*, que hereda la clase *Label* como se muestra en la figura 4.30, adicionalmente a los atributos heredados se asigna el atributo *name*.

```
34  class Label(LabelMixin):
35      meta = {'collection':'labels'}
36
37      name = StringField(required=True)
```

Figura 4.30 Clase *Label*
Fuente: Elaboración propia

Los atributos de la clase *Label* son almacenados en la colección *labels* dentro de *db_persea* y para el almacenamiento de información en la colección *labels_detail* se realiza el mismo proceso.

Rutas.

Las rutas del prototipo en *Flask* se encuentra en el directorio *apps* en el archivo *api.py*, mediante este archivo se puede acceder a las rutas definidas haciendo uso del *URL* en el navegador, estas rutas fueron utilizadas por *Frontend*. El enrutador en el prototipo utiliza *GET y POST*, para enviar peticiones al servidor *(request)* y recibir respuesta a la solicitud *(response)*.

```python
def configure_api(app):
    #addicionar ruta
    api.add_resource(Index,'/')
    #crear Etiqueta
    api.add_resource(LabelCreate,'/label')
    # update, get
    api.add_resource(LabelResources,'/label/<string:label_id>')
    # Listar
    api.add_resource(LabelList,'/lista_labels')
    # delete
    api.add_resource(LabelDelete,'/delete/<string:label_id>')


    #crear detalle etiqueta
    api.add_resource(LabelDetailCreate,'/label/detail')

    #listar id
    api.add_resource(LabelDetailList,'/label/details/<string:id_label>')
    #listar
    api.add_resource(DetailList,'/label/details')
    #update ,eliminar detalle etiqueta
    api.add_resource(LabelDetailResources,'/details/<string:id_label>')
    #upload Archivo
    api.add_resource(UploadPhoto,'/upload')
    #Prediccion
    api.add_resource(Prediccion,'/prediccion')

    #inicializacion de api
    api.init_app(app)
```

Figura 4.31 Rutas del Backend *api.py*
Fuente: Elaboración propia

La figura 4.31 muestra la declaración de la rutas utilizadas por el *backend*, por medio de los métodos de protocolo HTTP para la peticiones del cliente y sus respectivas respuestas, que son procesados en los controladores, cada ruta es procesada y se relaciona a una funcion implementada en los controladores.

Controladores.

Los controladores se ubican en el directorio *label* y *prediction* en los archivos *resources.py*. En los controladores se implementaron funciones que permiten manejar la lógica de la aplicación y trabajar con las peticiones *GET y POST*, vinculados a las rutas.

Para el prototipo se implementó la función de predicción haciendo uso de los modelos previamente entrenados, el proceso realizado es el siguiente:

- Al subir una imagen es almacenada en el directorio de nombre *tests* para la implementación del proceso se utiliza la función *UploadPhoto* como se muestra en la figura 4.32.

```python
class UploadPhoto(Resource):
    def post(self):
        parse = reqparse.RequestParser()
        parse.add_argument('file',
        type=werkzeug.datastructures.FileStorage,
        location='files')
        args = parse.parse_args()
        audioFile = args['file']
        audioFile.save("./apps/prediction/storage/tests/test.jpg")
```

Figura 4.32 Función *UploadPhoto* en archivo *resources.py*
Fuente: Elaboración propia

En *UploadPhoto* se define el método *POST*, esta función permite subir la imagen y guardarla en la carpeta test ubicada en el servidor para su posterior uso.

- Se realiza el preprocesado de la imagen para la predicción, en este paso la imagen que se encuentra en el servidor es normalizada mediante las librerías *opencv-python* y *numpy*.

```python
def preprocessing(filepath,size):
    test_img = []
    img_array = cv2.imread(filepath)
    img_resize = cv2.resize(img_array,(size,size))
    img_rgb = cv2.cvtColor(img_resize, cv2.COLOR_BGR2RGB)
    test_img.append(img_rgb)
    X = np.array(test_img)
    test_x = X/255.0
    return test_x
```

Figura 4.33 Función *preprocessing* en archivo *util.py*
Fuente: Elaboración propia

En la figura 4.33 se presenta la función *preprocessing*, recibe la ubicación de la imagen *(filepath)* y el tamaño *(size)* como parámetro, la ruta es leída y almacenada en la variable *img_array*, está pasa por el proceso de redimensionamiento y la aplicación de la escala de colores RGB, estos valores son almacenados en el arreglo *test_img*.

Para la normalización del arreglo *test_img* se dividen los valores entre 225 y son almacenados en un *Numpy Array*, con la finalidad de que los valores del arreglo se encuentren entre cero y uno, que se le asigna al arreglo *test_x* que retorna la función.

- La función de predicción permite identificar la deficiencia de macronutrientes que presenta la hoja mediante una imagen.

- El prototipo utiliza la clase *Predicción*, que se encuentra en el archivo *resources.py*, como se muestra en la figura 4.34.

```
18  class Prediccion(Resource):
19      def get(self):
20          predi = prediction()
21
22          return resp_ok(
23              'prediccion',MSG_RESOURCE_FETCHED.format('Predicciones '),
24              data=predi
25          )
```

Figura 4.34 Clase *Prediccion* en archivo *resources.py*
Fuente: Elaboración propia

La figura 4.34 muestra el uso de la función *prediction* que esta dentro del archivo *util.py*, donde se realiza el proceso de predicción.

```
30  def prediction(name):
31      # cargamos el model
32      ruta = os.getcwd()  + '/apps/prediction/storage'
33      model = load_model(ruta+ '/Modelo/' + name)
34      predi = model.predict(preprocessing(ruta + '/tests/test.jpg'))*100
35      print(predi)
36
37      return predi.tolist()
```

Figura 4.35 Función *prediction* en archivo *util.py*
Fuente: Elaboración propia

En la figura 4.35, el primer paso es ubicar el modelo dentro de la distribución de capetas haciendo uso de la librería *os*, para ser cargado por el método *load_model* que proporciona la librería *tesorflow* guardado en la variable *model*. La función *predict* propia de la variable model recibe como parámetro la imagen preprocesada multiplicado por 100. Esta función retorna un *Numpy Array* que asignado a la variable *predi* y finalmente se transforma la variable *predi* mediante la función *tolist* en una lista que será el valor que retorne la función *prediction*.

4.4.2. Frontend.

El *Frontend* se encarga de interactuar con el cliente el cual solicita *(request)* y obtiene una respuesta *(response)* por parte del servidor *(backend)*. Para desarrollar el *Frontend* se utilizó el *framework Angular 5*. El desarrollo de esta sección se centró en el módulo de predicción que se detalla más adelante.

Tabla 4.18 *Recursos empleados en desarrollo del frontend*

Framework	Angular 5
Librerias	Bootsrap
	jQuery

Fuente: Elaboración propia.

4.4.2.1. Estructura de *Frontend*.

El *Frontend* está conformado por una distribución de directorios que proporciona una estructura ordenada. El *framework* seleccionado para el desarrollo es *Angular 5*, que suministra módulos de *Nodejs* para ser utilizados en el proyecto. La figura 4.36 muestra la distribución de directorios dentro del *Frontend*.

En la figura 4.36 se encuentran agrupados archivos y directorios de la aplicación con bordes de color que se explicarón a continuación:

- ***Assets.*** Agrupado con borde de color celeste, dentro del directorio *assets*, se localiza las hojas de estilos *(CSS)*, las imágenes y archivos *JavaScript (Js)* que utiliza las plantillas *HTML* dentro del proyecto.

- ***Environments.*** Con borde de color azul, este directorio contiene los archivos *environment.ts* y *environment.prod.ts*, que son creados por defecto por el *framework* para declarar la ruta de conexión al *Backend*.

- ***Modules.*** Con borde color rojo, el archivo de nombre *app.modules.ts* permite declarar los componentes, los servicios e importar librerías necesarias para el desarrollo del proyecto, es importante por que unifica cada componente utilizado dentro del proyecto.

- ***Components.*** Con borde color azul royal tenemos *component.ts*, estos archivos son bloques de construcción básicas dentro de la aplicación, se relaciona con *service.ts*,

model.ts y plantillas HTML, empleando estos para crear funcionalidades como realizar solicitudes *(request)* y recibir las respuestas *(response)* que son usadas para crear los aspectos visuales y funcionalidades que se emplearán en el *Frontend*.

- ***Services.*** Los archivos *service.ts* de borde color verde, son utilizados para implementar métodos que realicen solicitudes haciendo uso de URL del protocolo HTTP que fueron creadas en el API y recibir las respuestas de estas, para ser utilizadas por el *component.ts*.
- ***Model.*** Con borde de color lila, los archivos *model.ts* son empleado para la creación de objetos que son utilizados en los *components.ts*.
- **Plantilla HTML.** Los archivos *label.html* y *prediction.html* con borde de color naranja, son utilizados para implementar la interfaz gráfica mediante funcionalidades asíncronas, vinculando atributos y métodos del componente con el que esté relacionado la plantilla HTML.
- ***Routing.*** Con borde color marrón el archivo *app.routing*, permite crear rutas URL del proyecto para acceder mediante el navegador.

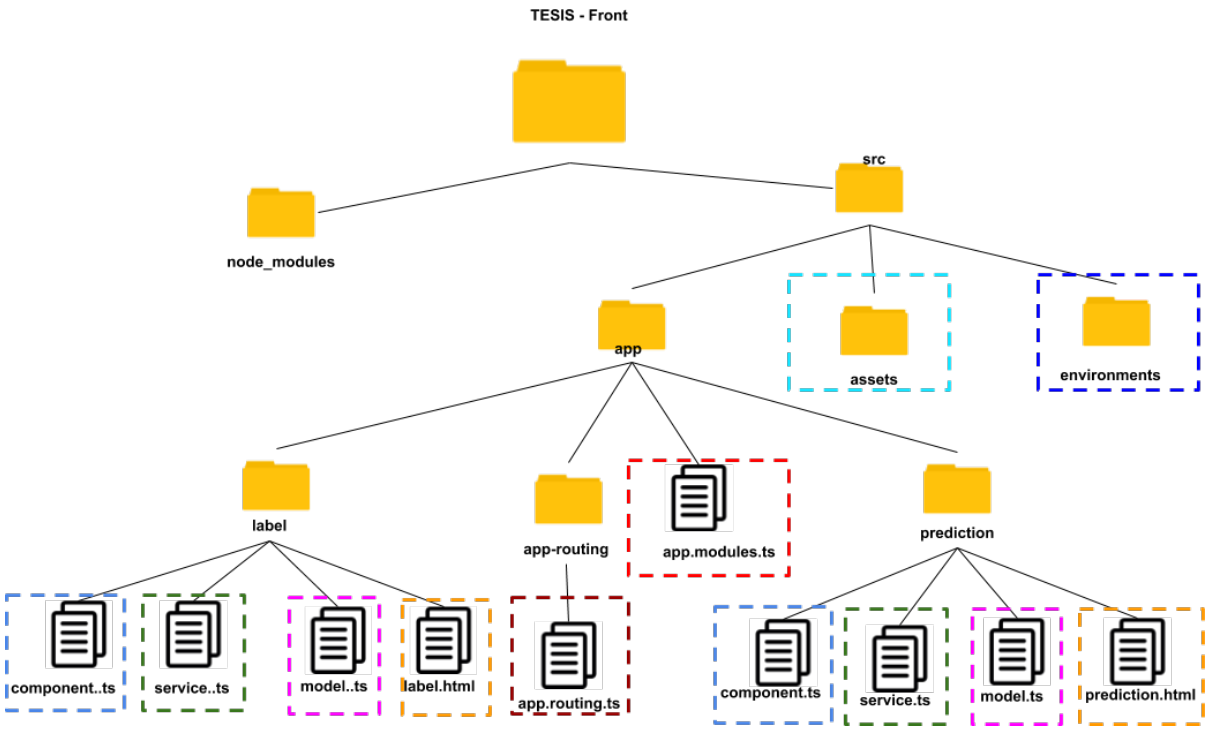


Figura 4.36 Estructura directorios y archivos del *Frontend*
Fuente: Elaboración propia

4.4.2.2. Interfaz gráfica de usuario.

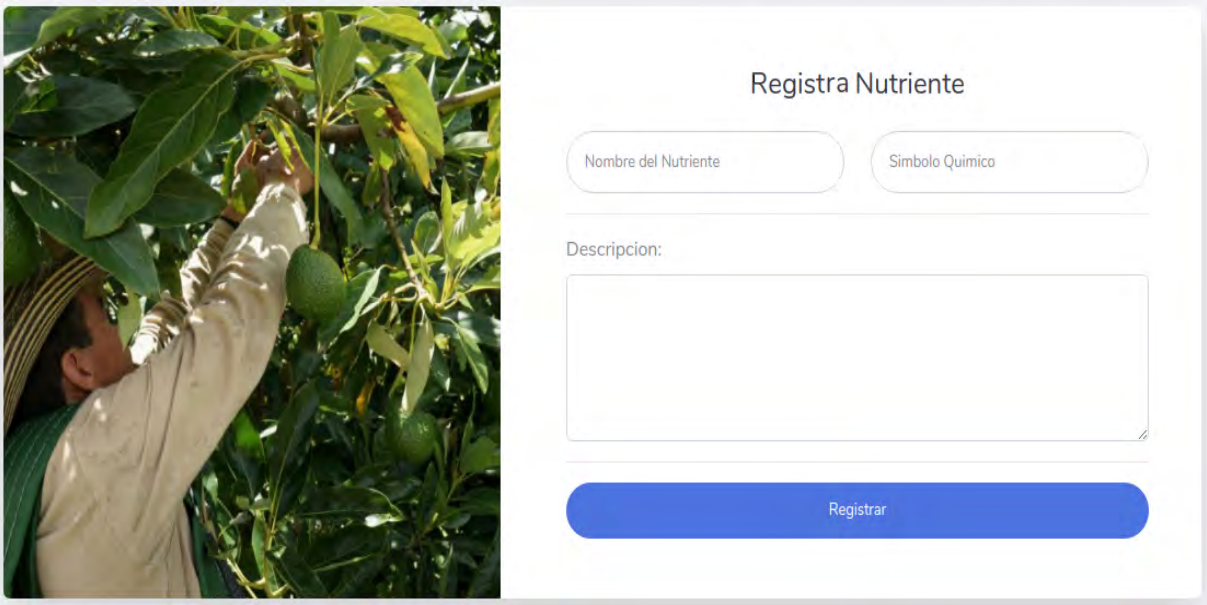
El GUI *(graphical user interface)* implementado para establecer comunicación entre el usuario y el programa. El GUI del proyecto fue desarrollado con framework *Angular 5*, *Bootstrap* y *JQuery*. La interfaz gráfica de proyecto comprende los siguientes módulos:

- **Presentación:** Es la primera pagina que se muestra cuando se ingresa al prototipo usando la dirección *URL* en el navegador, debido a que el prototipo funciona de manera local la ruta es la siguiente *http://localhost:4200/.*

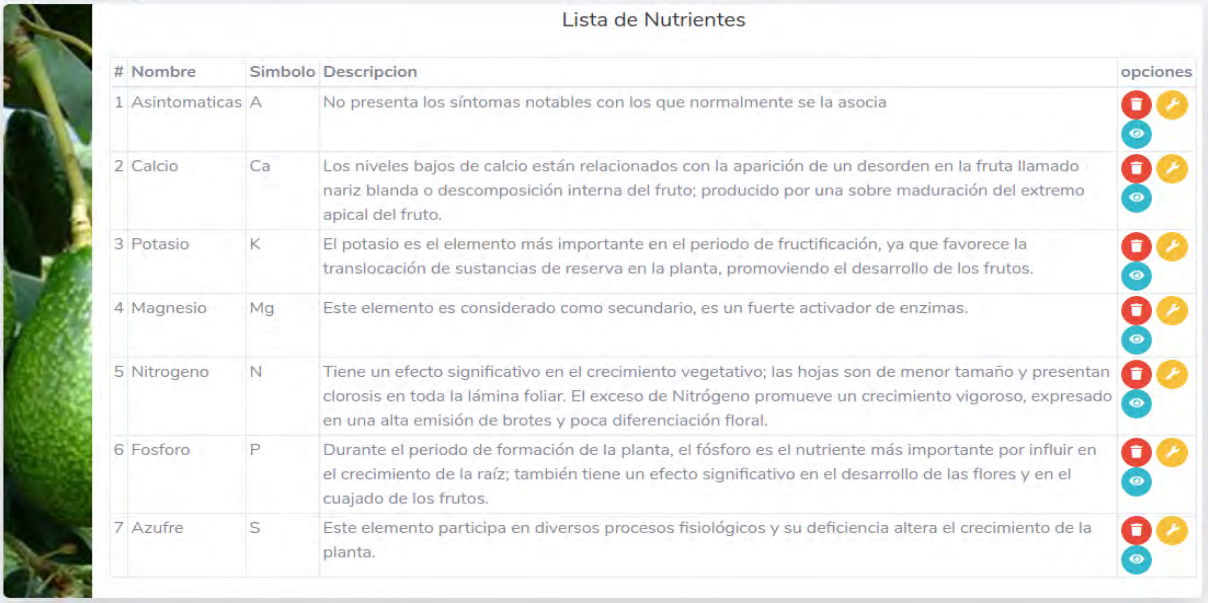
Figura 4.37 Modulo de Presentación
Fuente: Elaboración propia

En la figura 4.37 se observa al lado derecho la barra de menú, que da acceso a los módulos que contiene el sistema y al izquierdo la presentación.

- **Nutrientes:** En este módulo se implementó los *CRUD (Create, Read, Update and Delete)* de las *colecciones*, son los que almacenan la información de los macro nutrientes, las deficiencias y recomendaciones para el cuidado de la *Persea Americana*, este módulo se encarga de los mantenimientos del prototipo para ser utilizado por el módulo de predicción.



(a) Registrar nutriente



(b) Listar nutriente

Figura 4.38 Modulo registrar y listar nutriente
Fuente: Elaboración propia

En la figura 4.38(a) se observa el módulo *registrar nutriente*, para el funcionamiento de este se necesita registrar el *nombre*, *símbolo* y *descripción* del nutriente, que son campos requeridos. Una vez registrado el nutriente se presenta el módulo *listar nu-*

trientes, La figura 4.38(b) muestra una tabla con columnas de los atributos y opciones que contiene botones con iconos, que permiten al usuario *editar, eliminar* y *visualizar* los nutrientes. El prototipo cuenta con módulos *listar detalle nutriente* y *agregar detalle nutriente* en los que se implementaron similares funcionalidades.

- **Predicción:** En este módulo se presenta la función principal del proyecto, que es predecir la deficiencia de macronutrientes que se encuentra en las hojas de *Persea Americana*. La ruta URL para este proceso es *http://localhost:4200/admin/prediction*, mediante esta se accede al módulo de predicción, para el análisis de resultado se implementó gráficas de sectores circulares y barras, el módulo predice la deficiencia del macronutrientes utilizando tres arquitecturas *Inception-v3*, *DenseNet* y *MobileNet*.

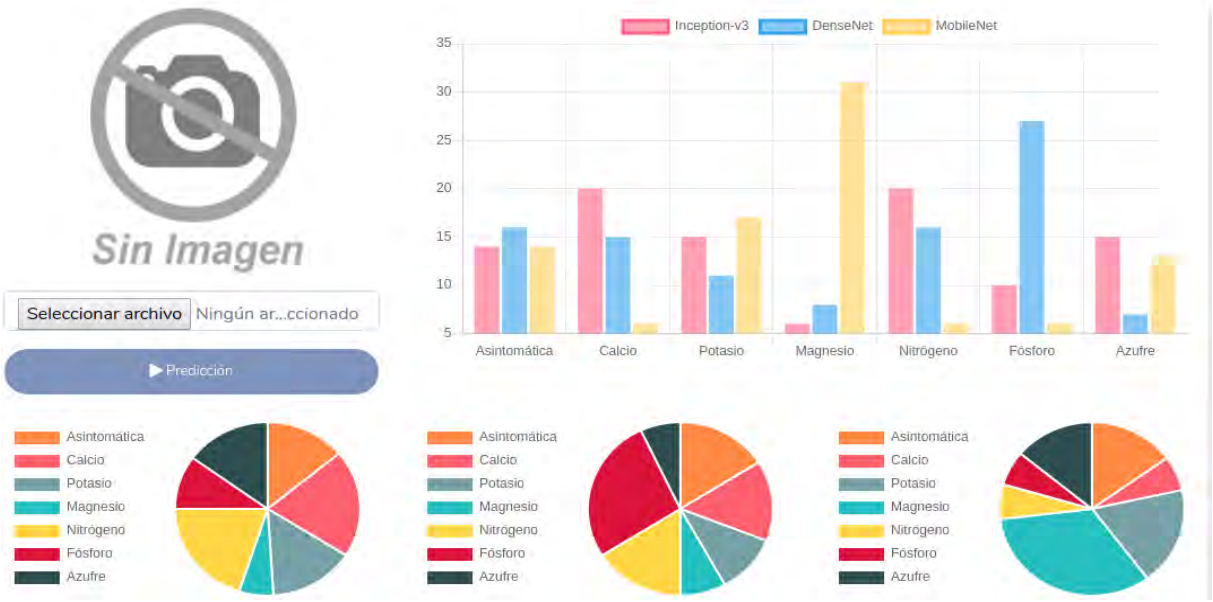
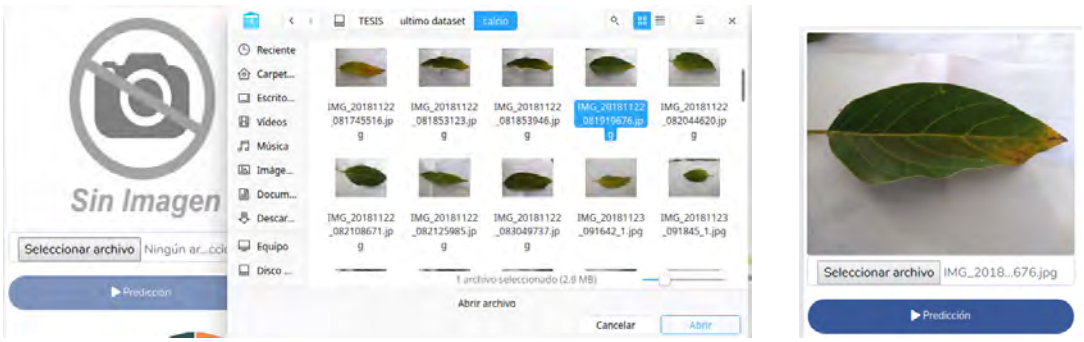


Figura 4.39 Modulo de Predicción
Fuente: Elaboración propia

La figura 4.39 muestra el inicio de este módulo, para realizar las pruebas será necesario seleccionar una imagen que corresponda a las características de la deficiencia de macronutrientes a predecir. Usando el botón *Seleccionar archivo* se carga la imagen ha ser usada, para iniciar con el proceso de predicción se presiona el botón *Predicción*.

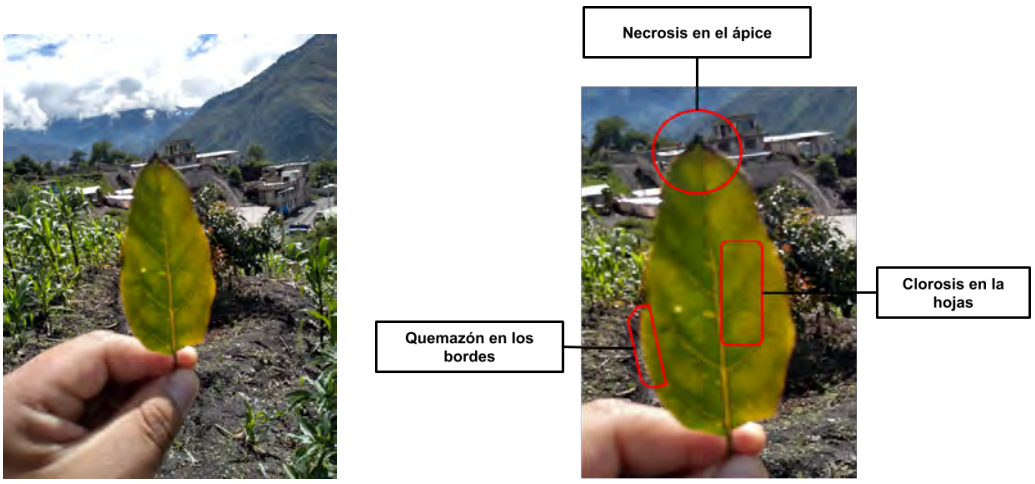


(a) Selección imagen (b) Predicción de imagen

Figura 4.40 Modulo Predicción de Nutriente
Fuente: Elaboración propia

4.4.2.3. Prueba de funcionamiento.

La principal funcionalidad del prototipo se presenta en el módulo *Predicción de nutriente*, las pruebas fueron realizadas en dicho módulo, debido que la orientación del proyecto es identificar las anomalías por deficiencia de macronutrientes que presenta la *Persea Americana* variedad *Hass*, caracterizadas en las hojas. Para realizar las pruebas se utilizó una imagen correspondiente a la deficiencia de calcio *(Ca)* como se muestra en la figura 4.41(a), que fue analizada por los modelos entrenados que corresponden a las arquitecturas *Inception-v3*, *DenseNet* y *MobileNet*.



(a) Deficiencia de calcio (b) Síntomas deficiencia calcio

Figura 4.41 Hoja de Persea Americana correspondiente a Ca
Fuente: Elaboración propia

La imagen utilizada para la predicción no pertenece a *Persea Americana Hass Dataset*, esta fue recolectada para su uso exclusivo en pruebas. Presenta las características como reducción de tamaño, quemazón en los bordes, clorosis y necrosis en el ápice de la hoja ver figura 4.41(b).

En la figura 4.42 se observa el módulo de *predicción*, una vez realizado la predicción de la imagen, el sistema proporciona información representada por medio de gráficas, permitiendo la comparación de los datos arrojados por los modelos de predicción.

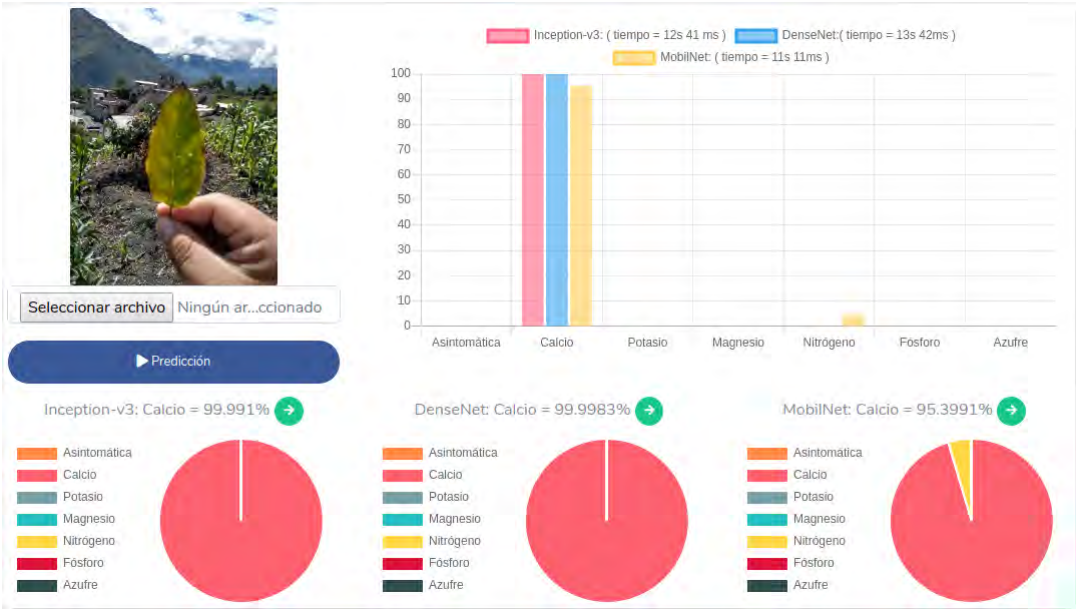


Figura 4.42 Modulo *predicción nutriente*
Fuente: Elaboración propia

El módulo presenta información del porcentaje de exactitud mediante una gráfica de sectores circulares, para cada modelo de predicción como se muestra en la figura 4.43, asimismo se presenta un gráfico de barra que hace una comparativa de los modelos utilizados, mediante el porcentaje de exactitud en función de los macronutriente como se muestra en la figura 4.44.

Figura 4.43 Gráfico de sectores circulares
Fuente: Elaboración propia

La figura 4.43 presentada en este módulo, es un gráfico que proporciona al usuario la interactividad, mostrando los porcentajes de exactitud en cada macronutriente.

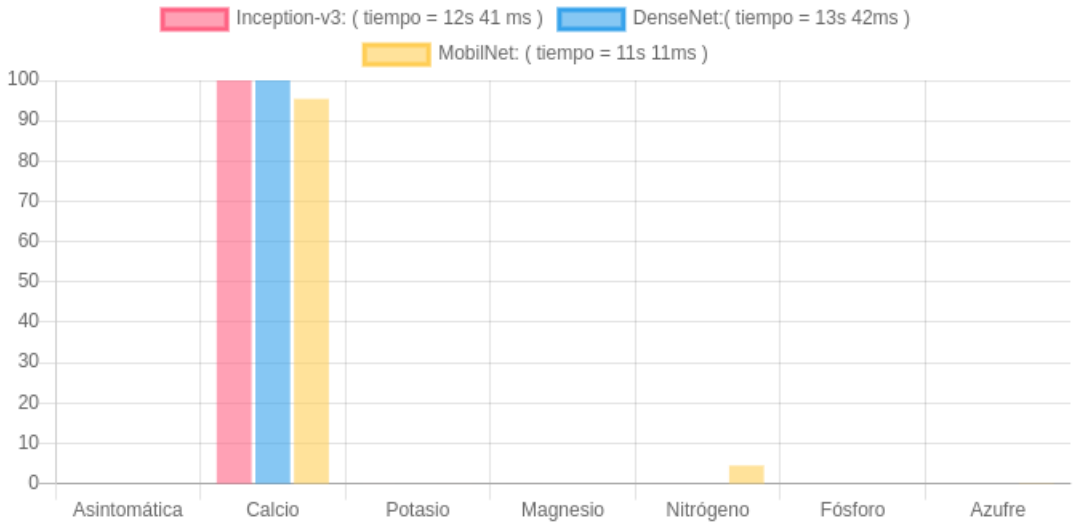


Figura 4.44 Gráfico de barras
Fuente: Elaboración propia

En la figura 4.44 se muestra una comparativa entre los modelos de predicción utilizados para el proyecto, siendo representados mediante gráfico de barras, asimismo se puede observar los tiempos de ejecución en el proceso de predicción para cada modelo, siendo al rededor de 55 segundos el tiempo total que tarda el prototipo en la predicción.

Capítulo 5

Resultados

5.1. Métricas de Evaluación

Finalizado el entrenamiento de las CNN construidas, el siguiente paso fue realizar la evaluación del desempeño aplicando métricas al subconjunto *valid* de *Persea Americana Hass Dataset* para el cálculo de los indicadores en cada modelo. Las métricas empleadas son:

- ***Accuracy* (Exactitud).** Fracción de predicciones que se realizaron correctamente en un modelo de clasificación. La exactitud se define de la siguiente manera:

$$Accuracy = \frac{Predicciones\ correctas}{Número\ total\ de\ ejemplos} \tag{5.1}$$

- ***Error rate* (Tasa de error).** Fracción de predicciones que se realizaron incorrectas en un modelo de clasificación. La tasa de error se define de la siguiente manera:

$$Error\ rate = \frac{Predicciones\ incorrectas}{Número\ total\ de\ ejemplos} \tag{5.2}$$

  El *Número total de ejemplos* es la suma de las predicciones correctas e incorrectas, de la suma de las ecuaciones 5.1 y 5.2 se obtiene:

$$Accuracy + Error\ rate = 1 \tag{5.3}$$

$$Error\ rate = 1 - Accuracy \tag{5.4}$$

- ***Precision* (Precisión).** La precisión identifica la frecuencia con la que un modelo predijo correctamente la clase positiva. Esto significa lo siguiente:

$$Precision = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ positivos} \tag{5.5}$$

- ***Recall*.** Indica la capacidad de detectar correctamente todas las etiquetas positivas posibles.

$$recall = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ negativos} \tag{5.6}$$

5.2. Clasificación de Macronutrientes

En está sección se evaluaron los modelos construidos a partir de las arquitecturas *Inception-v3*, *DenseNet169* y *MobileNet*, los resultados obtenidos de la evaluación de rendimiento provienen de las métricas aplicadas al subconjunto de validación.

5.2.1. Inception-v3.

Para esta arquitectura se implementaron tres modelos, la configuración de parámetros de entrenamiento se detalla en la tabla 5.1

Tabla 5.1 *Configuración de parámetros de entrenamiento de modelos Inception-v3.*

Nombre	Capas congeladas	Capas entrenadas	Épocas	Input
InceptionA	229	84	50	$229 \times 229 \times 3$
InceptionB	156	148	50	$229 \times 229 \times 3$
InceptionC	101	212	50	$229 \times 229 \times 3$

Fuente: Elaboración propia.

Resultados del entrenamiento.

Las gráficas 5.1, 5.2 y 5.3 muestran el comportamiento de la función *loss* y *accuracy* en los subconjuntos de *train* y *valid* durante las épocas de entrenamiento.

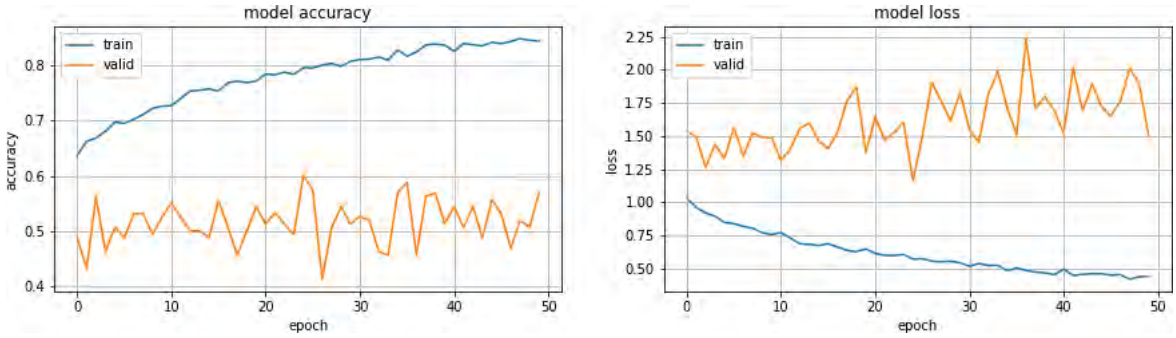


Figura 5.1 Gráfica de entrenamiento de InceptionA
Fuente: Elaboración propia.

El gráfico 5.1 corresponde al modelo *InceptionA* entrenado con 84 capas en 50 épocas, el entrenamiento tardó aproximadamente 3,100 s. Al lado derecho se observa resultados de la métrica *accuracy* con el subconjunto de entrenamiento que alcanzó 85 % con el subconjunto de entrenamiento y para el subconjunto de validación se alcanzó 60 %. Al lado izquierdo se observa el comportamiento de la función *loss*, para el subconjunto de entrenamiento decrece constantemente en contraste a *valid-loss* que está en constante crecimiento llegando a su pico alto entre las iteraciones 30 y 40.

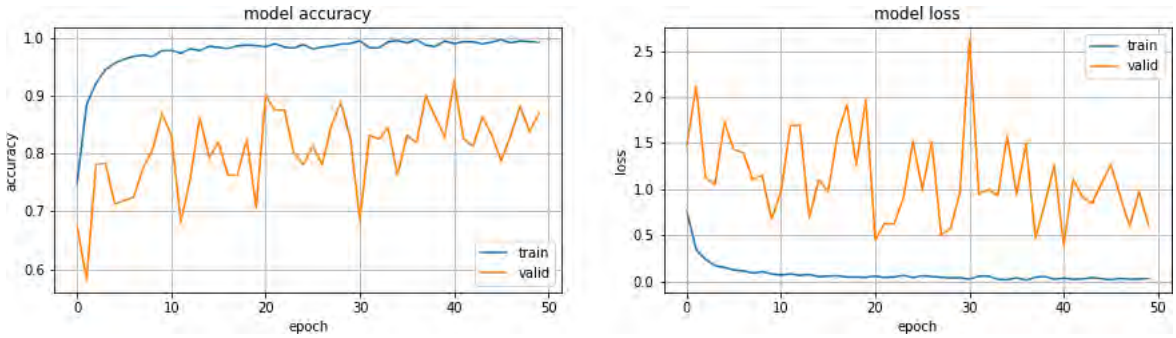


Figura 5.2 Gráfica de entrenamiento de InceptionB
Fuente: Elaboración propia.

El gráfico 5.2 corresponde al modelo *InceptionB* entrenado con 148 capas en 50 épocas, el entrenamiento tardó aproximadamente 3,200 s. De derecha a izquierda, se percibe los resultados de la métrica *accuracy* con el subconjunto de entrenamiento alcanzó 99.66 % y para el subconjunto de validación llega a un pico de 92.50 %. En cuanto a la función *loss* exhibe un comportamiento descendente en ambos subconjuntos. A comparación con el modelo anterior se infiere un mejor rendimiento.

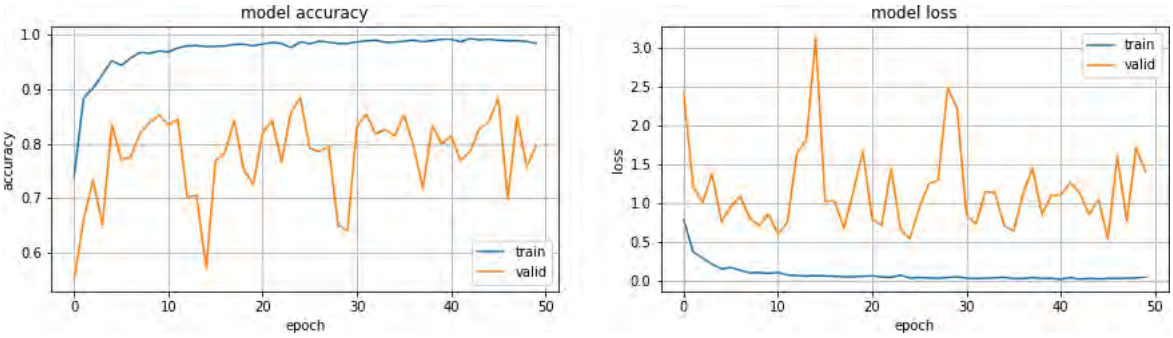
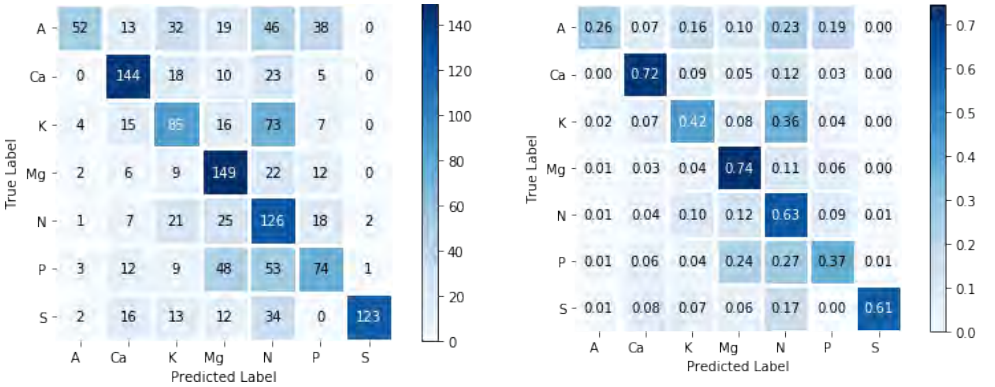


Figura 5.3 Gráfica de entrenamiento de InceptionC
Fuente: Elaboración propia.

El modelo *InceptionC* representado en la gráfica 5.3, entrenado con mayor profundidad con 212 capas en 50 épocas, tardó aproximadamente 3,650 s en el entrenamiento. Los resultados de la métrica *accuracy* con el subconjunto de entrenamiento alcanzó 99.25 % y el mayor porcentaje para el subconjunto de validación es de 88.45 %. En cuanto a la función *loss* el comportamiento del subconjunto de entrenamiento es descendente, a partir de la época 46 el valor crece. Mientras que con el subconjunto de validación los valores no bajan de 0.5.

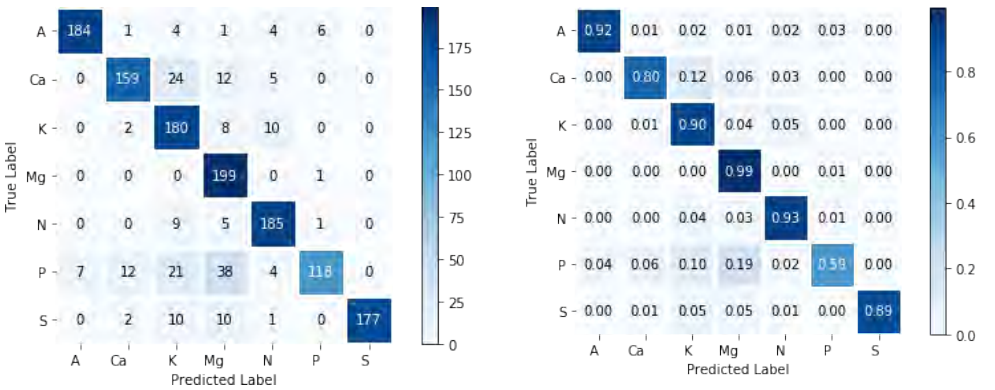
Evaluación de los modelos *Inception-v3*.

En las figuras 5.4 , 5.5 y 5.6 se muestran las matrices de confusión de las arquitectura *InceptionA*, *InceptionB* e *InceptionC* respectivamente. Cada figura contiene dos matrices, al lado izquierda se tiene la matriz de confusión sin normalizar y al lado derecho la matriz normalizada. En las matrices se observa la formación de la diagonal principal que indica el número de aciertos de cada una de las categorías del conjunto de datos.



(a) Matriz de confusión (b) Normalización de la matriz de confusión

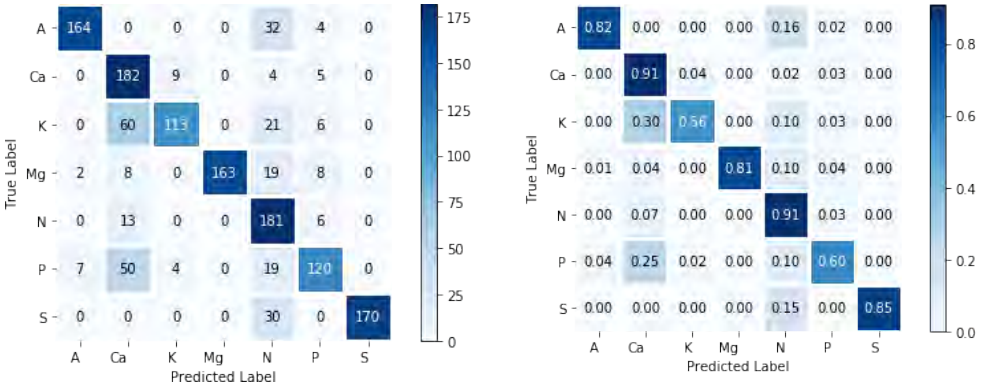
Figura 5.4 Matriz de confusión del modelo InceptionA



(a) Matriz de confusión (b) Normalización de la matriz de confusión

Figura 5.5 Matriz de confusión del modelo InceptionB

Fuente: Elaboración propia.



(a) Matriz de confusión (b) Normalización de la matriz de confusión

Figura 5.6 Matriz de confusión del modelo InceptionC
Fuente: Elaboración propia.

A partir de las matrices anteriores, se calcularon las métricas de evaluación por cada clase y se presenta la información en la tabla 5.2.

Tabla 5.2 *Métricas de los modelos Inception-v3.*

	InceptionA				***InceptionB***				***InceptionC***			
	# aciertos	acc	precision	recall	# aciertos	acc	precision	recall	# aciertos	acc	precision	recall
A	52	0.260	0.81	0.26	184	0.920	0.96	0.92	164	0.820	0.95	0.82
Ca	144	0.720	0.68	0.72	159	0.795	0.90	0.80	182	0.910	0.58	0.91
K	85	0.425	0.46	0.43	180	0.900	0.73	0.90	113	0.565	0.90	0.57
Mg	149	0.745	0.53	0.75	199	0.995	0.73	0.99	163	0.815	1.00	0.82
N	126	0.630	0.33	0.63	185	0.925	0.89	0.93	181	0.905	0.59	0.91
P	74	0.370	0.48	0.37	118	0.590	0.97	0.59	120	0.600	0.81	0.60
S	123	0.615	0.98	0.62	177	0.885	1.00	0.89	170	0.850	1.00	0.85

Fuente: Elaboración propia.

De la tabla 5.2 se observan los valores obtenidos en cada arquitectura *Inception-v3*:

- Del modelo *InceptionA*, la categoría **A** *(asintomáticas)* obtuvo 52 aciertos siendo la más baja, adicionalmente las clases **P** *(fósforo)* y **K** *(potasio)* también tienen una baja clasificación de 74 y 85 aciertos respectivamente. Mientras que la clase **Mg** *(magnesio)* tiene 149 aciertos de las 200 imágenes que conforman el subconjunto de validación para esta categoría.
- Para el modelo *InceptionB*, la clase con menor aciertos es **P** *(fósforo)* que predijo 118 imágenes correctamente y **Mg** *(magnesio)* obtuvo la mayor cantidad de acierto. El comportamiento del modelo para las otras categorías alcanzó un buen desempeño a diferencia del modelo anterior.

- Las clases **K** *(potasio)* y **P** *(fósforo)* tienen una menor cantidad de aciertos al ser evaluadas en el modelo *InceptionC*, en contraste la categoría **Ca** *(calcio)* obtuvo el mayor número de predicciones correctas.

Al revisar los resultados de los tres modelos, se observa que la clase **Mg** *(magnesio)* es la que alcanzó mayor clasificación, a diferencia de la clase **P** *(fósforo)*. En la tabla 5.3 se sintetiza la información para analizar el rendimiento global de cada uno de los modelos.

Tabla 5.3 *Comparativa entre modelos InceptionA, InceptionB y InceptionC.*

	accuracy(%)	error rate (%)	precision
InceptionA	53.78	46.21	0.61
InceptionB	85.86	14.14	0.88
InceptionC	78.07	21.93	0.83

Fuente: Elaboración propia.

En la tabla 5.3 se exponen los resultados de la clasificación que realizan los tres modelos de *Inception* sobre las imágenes del subconjunto de validación. La arquitectura *InceptionB* obtuvo una tasa de acierto de 88.56 % y una tasa de error de 14.14 %, siendo este el modelo con mejor rendimiento.

5.2.2. DenseNet169.

Se implementaron tres modelos de está arquitectura, la configuración de parámetros de entrenamiento se detalla en la tabla 5.4

Tabla 5.4 *Configuración de parámetros de entrenamiento de modelos DenseNet.*

Nombre	Capas congeladas	Capas entrenadas	Épocas	Input
DenseNetA	474	123	20	$224 \times 224 \times 3$
DenseNetB	368	229	20	$224 \times 224 \times 3$
DenseNetC	274	323	20	$224 \times 224 \times 3$

Fuente: Elaboración propia.

Resultados del entrenamiento.

Las gráficas 5.7, 5.8 y 5.9 muestran el comportamiento de la función *loss* y *accuracy* en los subconjuntos de *train* y *valid* durante las iteraciones de entrenamiento.

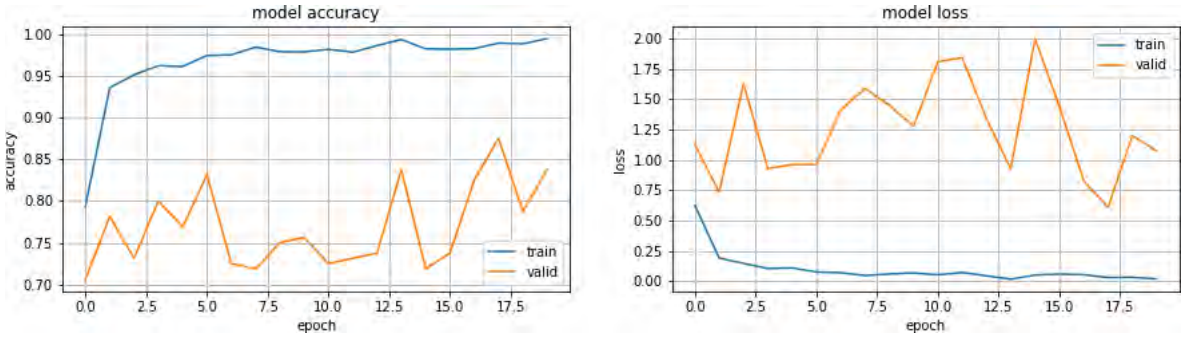


Figura 5.7 Gráfica de entrenamiento de DenseNetA
Fuente: Elaboración propia.

El gráfico 5.7 corresponde al modelo *DenseNetA* entrenado con 123 capas en 20 épocas, el entrenamiento tardó aproximadamente 1,735 s. De derecha a izquierda, se percibe los resultados de la métrica *accuracy* con el subconjunto de entrenamiento alcanzó 99.44 % y para el subconjunto de validación llega a un pico de 87.50 %. En cuanto a la función *loss* exhibe un comportamiento descendente en el conjunto de datos *train* y los datos valid decrece hasta 0.609.

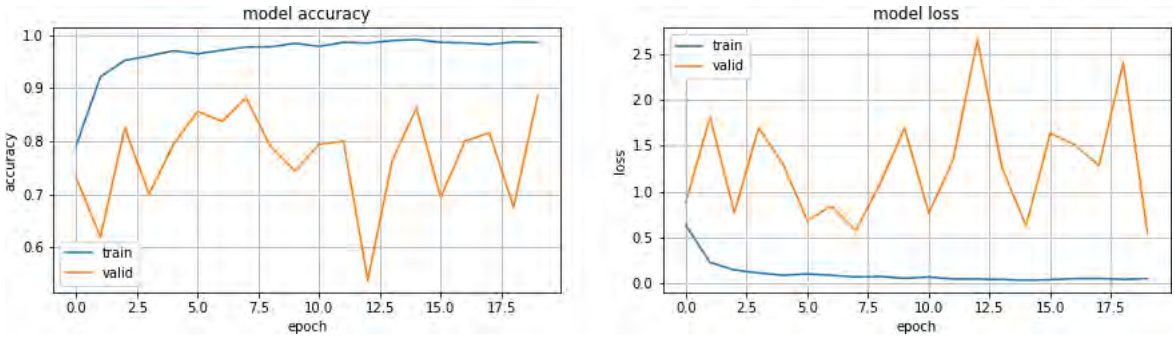


Figura 5.8 Gráfica de entrenamiento de DenseNetB
Fuente: Elaboración propia.

El modelo *DenseNetB* representado en la gráfica 5.8, entrenado con mayor profundidad con 229 capas en 20 épocas, el entrenamiento tardó aproximadamente 1,837 s. Los resultados de la métrica *accuracy* con el subconjunto de entrenamiento alcanzó 98.97 % y el mayor porcentaje para el subconjunto de validación es de 88.75 %. En cuanto a la función *loss* el comportamiento del subconjunto de entrenamiento es descendente y con el subconjunto de validación los valores no bajan de 0.57.

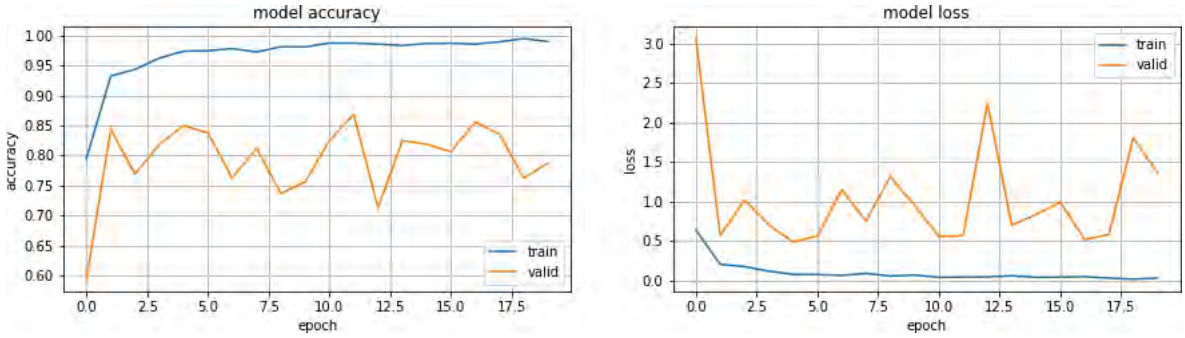


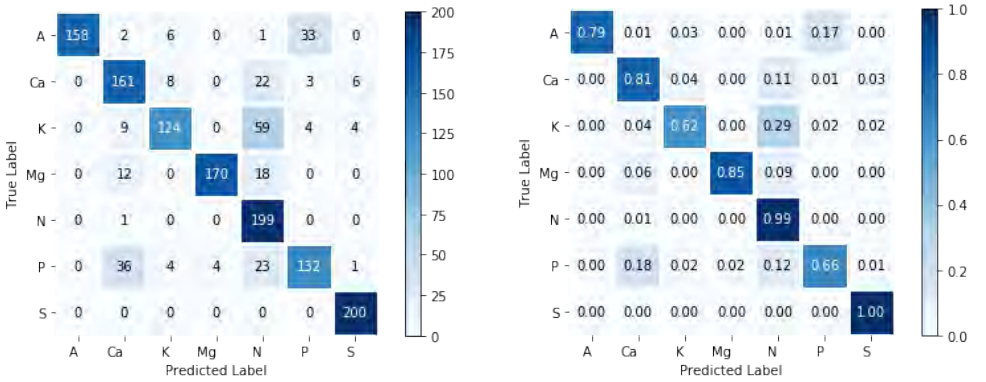
Figura 5.9 Gráfica de entrenamiento de DenseNetC

Fuente: Elaboración propia.

El gráfico 5.9 corresponde al modelo *DenseNetC* entrenado con 323 capas en 20 épocas, el entrenamiento tardó aproximadamente 1,956 s. Al lado derecho se observa resultados de la métrica *accuracy* con el subconjunto de entrenamiento se logró 99.51 % y para el subconjunto de validación alcanza el 86.88 %. Al lado izquierdo la función *loss* para el subconjunto de entrenamiento hasta 0.0155 y el conjunto *valid* tiene como mínimo valor 0.5938.

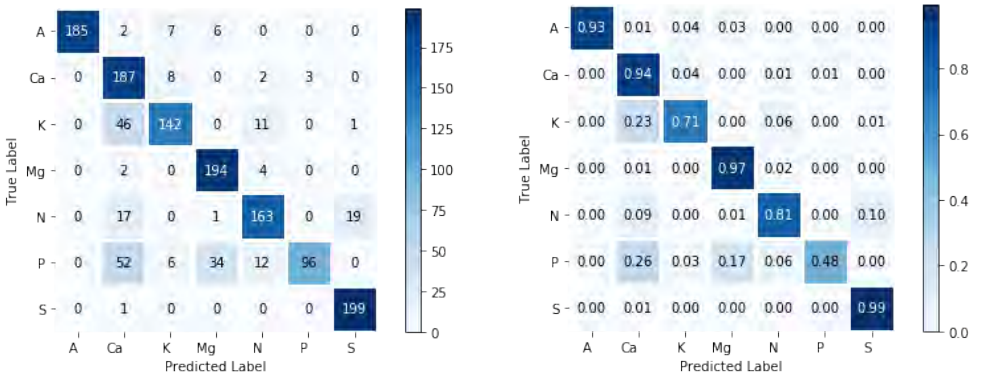
Evaluación de los modelos DenseNet.

En las figuras 5.10 , 5.11 y 5.12 se muestran las matrices de confusión de las arquitectura *DenseNetA*, *DenseNetB* y *DenseNetC* respectivamente. Cada figura contiene dos matrices al lado izquierdo se tiene la matriz de confusión sin normalizar y al lado derecho la matriz normalizada. En las matrices se observa la formación de la diagonal principal que indica el número de aciertos de cada una de las categorías del conjunto de datos.



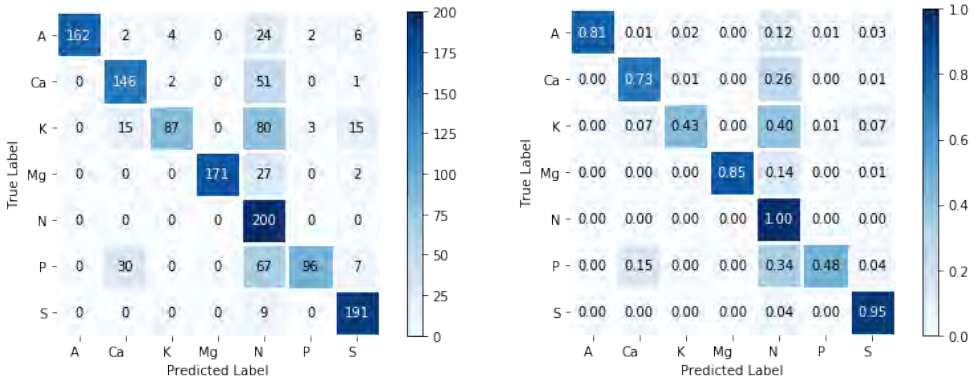
(a) Matriz de confusión (b) Normalización de la matriz de confusión

Figura 5.10 Matriz de confusión del modelo DenseNetA
Fuente: Elaboración propia.



(a) Matriz de confusión (b) Normalización de la matriz de confusión

Figura 5.11 Matriz de confusión del modelo DenseNetB
Fuente: Elaboración propia.



(a) Matriz de confusión (b) Normalización de la matriz de confusión

Figura 5.12 Matriz de confusión del modelo DenseNetA
Fuente: Elaboración propia.

A partir de las matrices anteriores, se calcularon las métricas de evaluación por cada clase y se presenta la información en la tabla 5.5.

Tabla 5.5 *Métricas de los modelos DenseNet.*

	DenseNetA				***DenseNetB***				***DenseNetC***			
	# aciertos	acc	precision	recall	# aciertos	acc	precision	recall	# aciertos	acc	precision	recall
A	158	0.790	1.00	0.79	185	0.925	1.00	0.93	162	0.810	1.00	0.81
Ca	161	0.805	0.73	0.81	187	0.935	0.61	0.94	146	0.730	0.76	0.73
K	124	0.620	0.87	0.62	142	0.710	0.87	0.71	87	0.435	0.94	0.44
Mg	170	0.850	0.98	0.85	194	0.970	0.83	0.97	171	0.855	1.00	0.86
N	199	0.995	0.62	1.00	163	0.815	0.85	0.82	200	1.00	0.44	1.00
P	132	0.660	0.77	0.66	96	0.480	0.97	0.48	96	0.480	0.95	0.48
S	200	1.00	0.95	1.00	199	0.995	0.91	1.00	191	0.995	0.86	0.96

Fuente: Elaboración propia.

De la tabla 5.5 se observan los valores obtenidos en cada arquitectura DenseNet:

- Para el modelo *DenseNetA*, la clase con menor aciertos es **K** *(potasio)* que predijo 124 imágenes correctamente y **S** *(azufre)* obtuvo el mayor número de predicciones correctas.
- Las clases **K** *(potasio)* y **P** *(fósforo)* tienen una menor cantidad de aciertos al ser evaluadas en el modelo *DenseNetB*, en contraste la categoría **S** *(azufre)* obtuvo el mayor número de predicciones correctas.
- Del modelo *DenseNetC*, la categoría **K** *(potasio)* obtuvo 87 aciertos siendo la más baja, adicionalmente la clase **P** *(fósforo)* tiene una baja clasificación de 96 aciertos.

Mientras que la clase **N** *(nitrógeno)* tiene 200 aciertos de las 200 imágenes que conforman el subconjunto de validación para está categoría.

Al revisar los resultados de los tres modelos, se observa que la clase **S** *(azufre)* alcanzó mayor clasificación, a diferencia de la clase **P** *(fósforo)*. En la tabla 5.6 se ha sintetizado la información para analizar el rendimiento global de cada uno de los modelos.

Tabla 5.6 *Comparativa entre modelos DenseNetA, DenseNetB y DenseNetC.*

	accuracy(%)	error rate (%)	precision
DenseNetA	81.72	18.29	0.84
DenseNetB	83.29	16.71	0.86
DenseNetC	75.21	24.78	0.85

Fuente: Elaboración propia.

En la tabla 5.6 se exponen los resultados de la clasificación que realizan los tres modelos de *DenseNet* sobre las imágenes del subconjunto de validación. La arquitectura *DenseNetB* obtuvo una tasa de acierto de 83.29 % y una tasa de error de 16.71 %, siendo este el modelo con mejor rendimiento.

5.2.3. MobileNet.

Se implementaron tres modelos de está arquitectura, la configuración de parámetros de entrenamiento se detalla en la tabla 5.7

Tabla 5.7 *Configuración de parámetros de entrenamiento de modelos MobileNet.*

Nombre	Capas congeladas	Capas entrenadas	Épocas	Input
MobileNetA	60	30	50	$224 \times 224 \times 3$
MobileNetB	40	50	50	$224 \times 224 \times 3$
MobileNetC	20	70	50	$224 \times 224 \times 3$

Fuente: Elaboración propia.

Resultados del entrenamiento.

Las gráficas 5.13, 5.14 y 5.15 muestran el comportamiento de la función *loss* y *accuracy* en los subconjuntos de *train* y *valid* durante las iteraciones de entrenamiento.

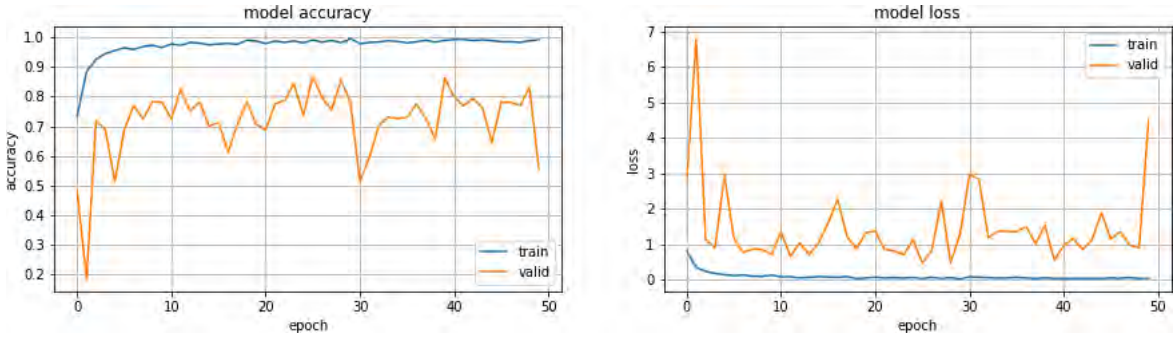


Figura 5.13 Gráfica de entrenamiento de MobileNetA
Fuente: Elaboración propia.

El gráfico 5.13 corresponde al modelo *MobileNetA* entrenado con 30 capas en 50 épocas, el entrenamiento tardó aproximadamente 1,155 s. Se observa resultados de la métrica *accuracy*, que con el subconjunto de entrenamiento logró 99.72 % y para el subconjunto de validación alcanza el 88.12 %. La función *loss* exhibe un comportamiento descendente en ambos subconjuntos y una elevación cerca de la iteración 50.

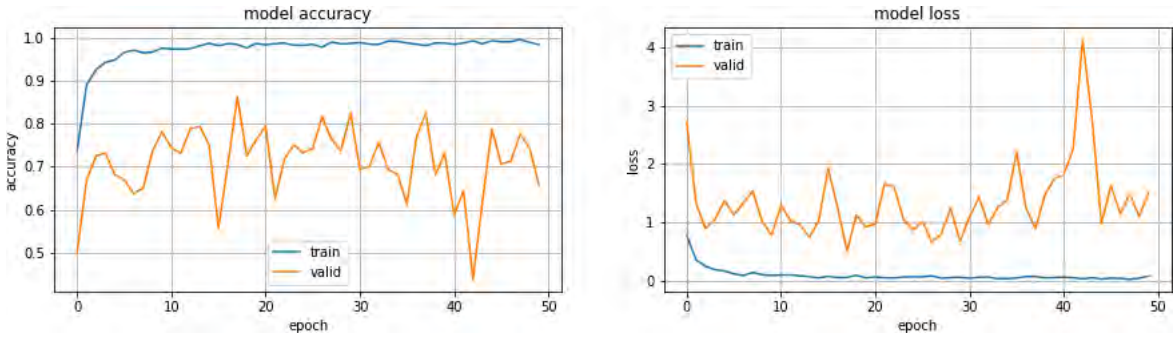


Figura 5.14 Gráfica de entrenamiento de MobileNetB
Fuente: Elaboración propia.

El gráfico 5.14 corresponde al modelo *MobileNetB* entrenado con 50 capas en 50 épocas, el entrenamiento tardó aproximadamente 1,509 s. De derecha a izquierda, se percibe los resultados de la métrica *accuracy* con el subconjunto de entrenamiento que alcanzó 99.59 % y para el subconjunto de validación llega a un pico de 86.18 %. En cuanto a la función *loss* el comportamiento de datos de entrenamiento es descendente y la validación toma valores entre 4.1377 máximo a 0.5028 mínimo.

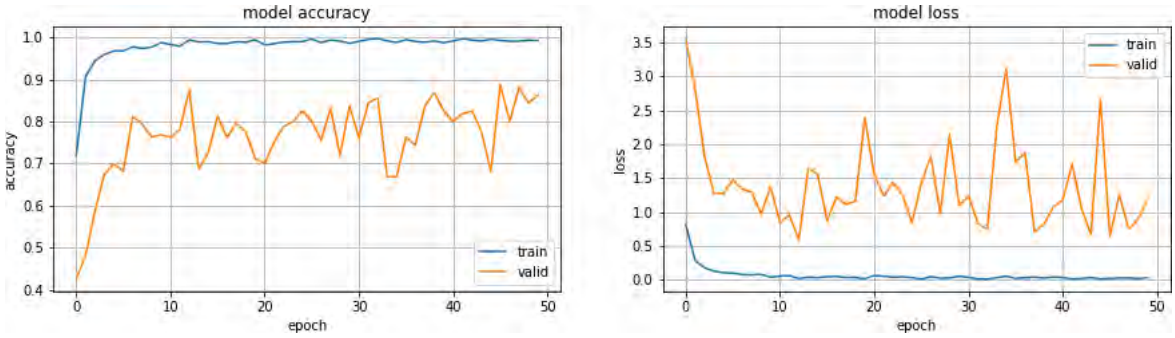
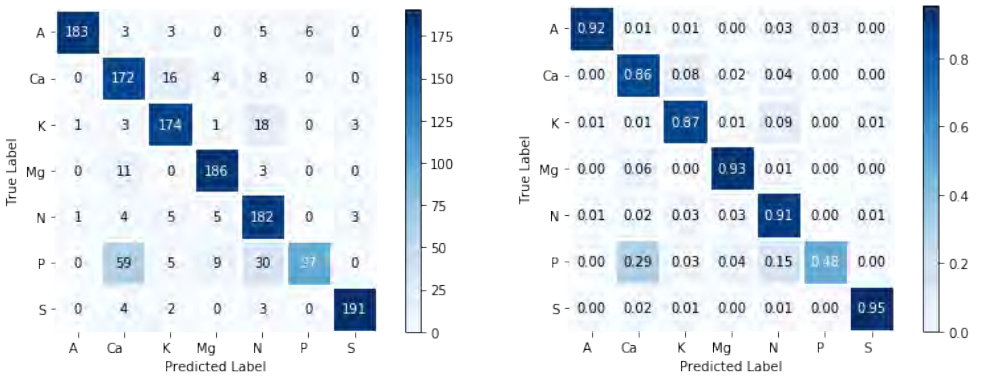


Figura 5.15 Gráfica de entrenamiento de MobileNetC
Fuente: Elaboración propia.

El modelo *MobileNetC* representado en la gráfica 5.15, entrenado con mayor profundidad con 70 capas en 50 épocas, el entrenamiento tardó aproximadamente 1,008 s. Los resultados de la métrica *accuracy* con el subconjunto de entrenamiento alcanzó 99.68 % y el mayor porcentaje para el subconjunto de validación es de 86.88 %. En cuanto a la función *loss* el comportamiento del subconjunto de entrenamiento es descendente. Mientras que con el subconjunto de validación los valores no bajan de 0.5.

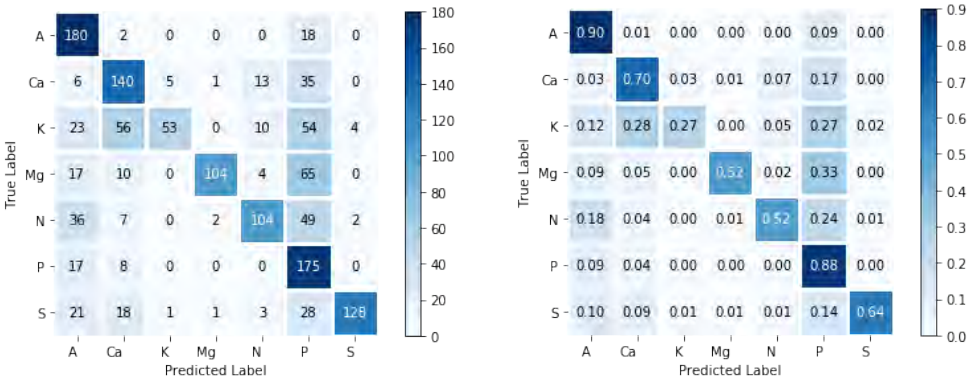
Evaluación de los modelos MobileNet.

En las figuras 5.16 , 5.17 y 5.18 se muestran las matrices de confusión de las arquitecturas *MobileNetA*, *MobileNetB* y *MobileNetC* respectivamente. Cada figura contiene dos matrices, al lado izquierdo se tiene la matriz de confusión sin normalizar y al lado derecho la matriz normalizada. En las matrices se observa la formación de la diagonal principal que indica el número de aciertos de cada una de las categorías del conjunto de datos.



(a) Matriz de confusión (b) Normalización de la matriz de confusión

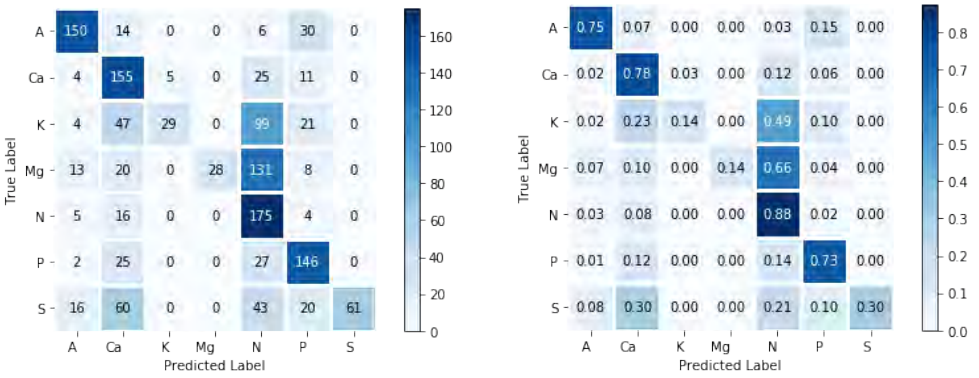
Figura 5.16 Matriz de confusión del modelo MobileNetA



(a) Matriz de confusión (b) Normalización de la matriz de confusión

Figura 5.17 Matriz de confusión del modelo MobileNetB

Fuente: Elaboración propia.



(a) Matriz de confusión (b) Normalización de la matriz de confusión

Figura 5.18 Matriz de confusión del modelo MobileNetC

Fuente: Elaboración propia.

A partir de las matrices anteriores, se calcularon las métricas de evaluación por cada clase y se presenta la información en la tabla 5.8.

Tabla 5.8 *Métricas de los modelos MobileNet.*

	MobileNetA				***MobileNetB***				***MobileNetC***			
	# aciertos	acc	precision	recall	# aciertos	acc	precision	recall	# aciertos	acc	precision	recall
A	183	0.915	0.99	0.92	180	0.900	0.60	0.90	150	0.750	0.77	0.75
Ca	172	0.860	0.67	0.86	140	0.700	0.58	0.70	155	0.775	0.46	0.78
K	174	0.870	0.85	0.87	53	0.265	0.90	0.27	29	0.145	0.85	0.15
Mg	186	0.930	0.91	0.93	104	0.520	0.96	0.52	28	0.140	1.00	0.14
N	182	0.910	0.73	0.91	104	0.520	0.78	0.52	175	0.875	0.35	0.88
P	97	0.485	0.94	0.49	175	0.875	0.41	0.88	146	0.730	0.61	0.73
S	191	0.955	0.97	0.96	128	0.640	0.96	0.64	161	0.305	1.00	0.31

Fuente: Elaboración propia.

De la tabla 5.8 se observan los valores obtenidos en cada arquitectura *MobileNet*:

- Para el modelo *MobileNetA*, la clase con menor aciertos es **P** *(fósforo)* que predijo 97 imágenes correctamente y **S** *(azufre)* obtuvo el mayor número de predicciones correctas.

- Las clases **K** *(potasio)*, **Mg** *(magnesio)* y **N** *(nitrógeno)* tienen una menor cantidad de aciertos al ser evaluadas en el modelo MobileNetB, en contraste la categoría **A** *(asintomáticas)* obtuvo el mayor número de predicciones correctas.

- Del modelo *MobileNetC*, la categoría **Mg** *(magnesio)* obtuvo 28 aciertos siendo la más baja, seguido de **K *(potasio)*** con 29 aciertos. Mientras que la clase **N** *(nitrógeno)* tiene 175 aciertos de las 200 imágenes que conforman el subconjunto de validación para está categoría.

Al revisar los resultados de los tres modelos, se observa que la clase **A** *(asintomática)* alcanzó mayor clasificación, a diferencia de la clase **K** *(potasio)*. En la tabla 5.9 se ha sintetizado la información para analizar el rendimiento global de cada uno de los modelos.

Tabla 5.9 *Comparativa entre modelos MobileNetA, MobileNetB y MobileNetC.*

	accuracy(%)	error rate (%)	precision
MobileNetA	84.64	15.36	0.87
MobileNetB	63.14	36.86	0.74
MobileNetC	53.14	46.85	0.72

Fuente: Elaboración propia.

En la tabla 5.9 se exponen los resultados de la clasificación que realizan los tres modelos de *MobileNet* sobre las imágenes del subconjunto de validación. La arquitectura *MobileNetA* obtuvo una tasa de acierto de 84.64 % y una tasa de error de 15.36 %, siendo este el modelo con mejor rendimiento.

5.3. Comparativa de Resultados Obtenidos de los Modelos

Con la finalidad de realizar una comparativa del desempeño global de los modelos en la *Persea Americana Hass Dataset*. La tabla 5.10 muestra el resumen de los resultados presentados en las tablas 5.3, 5.6 y 5.9, donde se seleccionó al modelo que tiene el mejor promedio del indicador *accuracy*.

Tabla 5.10 *Comparativa entre modelos InceptionB, DenseNetB y MobileNetA.*

	accuracy(%)	error rate (%)	precision
InceptionB	85.86	14.14	0.88
DenseNetB	83.29	16.71	0.86
MobileNetA	84.64	15.36	0.87

Fuente: Elaboración propia.

De la información contenida en la tabla 5.10 se puede deducir:

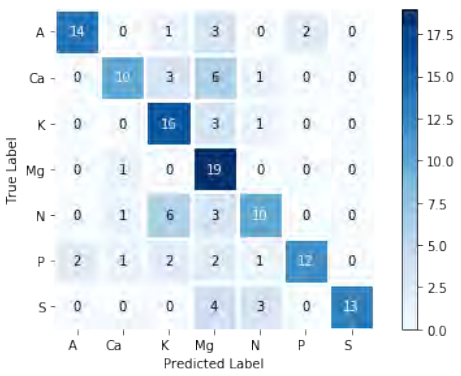
- El modelo *InceptionB* logró la mayor tasa de acierto de 85.86 % entre los tres modelos, mientras que el modelo *DenseNetB* posee la menor tasa de acierto de 83.29 % y el porcentaje de la tasa de acierto obtenido por *MobileNetA* hace que se posicione en el segundo lugar con 84.64 %.

- En consecuencia las tasas de error de los modelos *InceptionB, DenseNetB y MobileNetA* son 14.14 %, 16.71 % y 15.36 % respectivamente. La relación entre la tasa de acierto y tasa de error es inversamente proporcional.

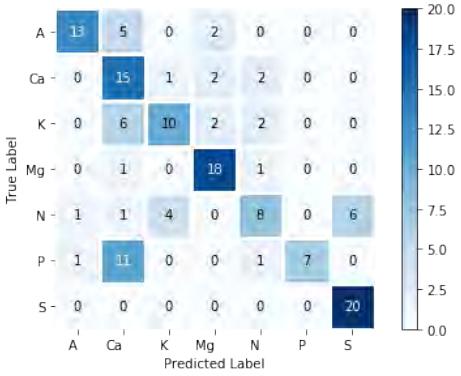
La tasa de error y tasa de acierto son indicadores significativos para evaluar el rendimiento de los modelos en la tarea de identificación de macronutrientes en la *Persea Americana Hass.*

5.4. Extrapolación de Resultados

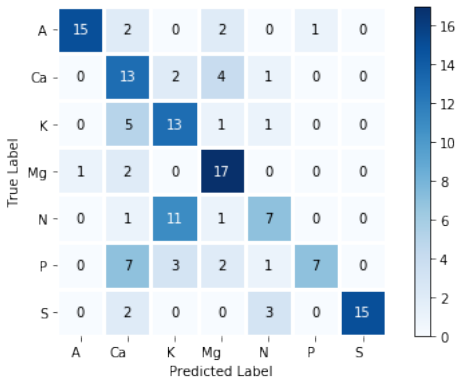
En está sección se evaluó el conjunto denominado test, conformado por 20 imágenes de cada categoría, que no están incluidas en *Persea Americana Hass Dataset*. Para está evaluación se emplearon las arquitecturas *InceptionB*, *DenseNetB* y *MobileNetA*, obteniendo las siguientes matrices:



(a) InceptionB



(b) DenseNetB



(c) MobileNetA

Figura 5.19 Matrices de confusión del subconjunto *test*
Fuente: Elaboración propia.

La diagonal principal de cada matriz representa la cantidad de aciertos obtenido por un modelo. Se sintetiza está información en la tabla 5.11

Tabla 5.11 *Resultados del subconjunto de test.*

	accuracy(%)	precision
InceptionB	67.14	0.74
DenseNetB	65.00	0.72
MobileNetA	62.14	0.69

Fuente: Elaboración propia.

De la información contenida en la tabla 5.11 se puede interpretar que el modelo InceptionB obtuvo un tasa de acierto del 67.14 % y continúa siendo la arquitectura con mejor desempeño en la tarea de identificación de anomalías por deficiencia de macronutrientes.

Conclusiones

1. El modelo construido a partir de la arquitectura *Inception-v3* demostró un mejor desempeño para la identificación de deficiencias de macronutrientes en la Persea Americana variedad Hass. Las predicciones realizadas por las arquitecturas MobileNet y DenseNet169, consiguieron tasas de acierto próximos a la alcanzada por Inception-v3.

2. La aplicación de transferencia de aprendizaje para conjuntos de datos pequeños como el recolectado para el trabajo de investigación *Persea Americana Hass Dataset*, fue esencial para el entrenamiento y derivo en modelos con buen desempeño en la identificación de deficiencias de macronutrientes.

3. Se elaboró una guía con las características de deficiencias de macronutrientes presentes en las hojas de Persea Americana variedad Hass, empleado para la recolección del *dataset*.

4. Se construyó el conjunto de datos *Persea Americana Hass Dataset* [1] con 6,090 imágenes que corresponden a siete categorías: Asintomáticas, Calcio, Potasio, Magnesio, Nitrógeno, Fósforo y Azufre. Las imágenes que conforman el *dataset* fueron recolectadas en el sector de Molinopata, provincia de Abancay, región Apurímac.

5. Se emplearon tres configuraciones de parámetros para cada una de las arquitecturas *Inception-v3*, *DenseNet169* y *MobileNet* aplicando la transferencia de aprendizaje, con la finalidad de lograr un mejor desempeño en la clasificación del macronutriente que genera anomalías en las hojas de la Persea Americana variedad Hass.

6. De los modelos implementados para las arquitecturas *Inception-v3*, *DenseNet169* y *MobileNet*, se alcanzaron los siguientes resultados en la identificación de anomalías generadas por deficiencia de macronutrientes en la Persea Americana variedad Hass:

[1] Dataset disponible en el siguiente enlace: *https://gitlab.com/betzabe/dshass.*

- De las configuraciones implementadas en *DenseNet169*, el modelo con 368 capas congeladas y 229 capas entrenadas en 20 épocas alcanza una tasa de acierto de 83.29 %, tasa de error de 16.71 % y una precisión de 0.86.
- Para la arquitectura *MobileNet*, el modelo con 60 capas congeladas y 30 capas entrenadas en 50 épocas alcanza una tasa de acierto de 84.64 %, tasa de error de 15.36 % y una precisión de 0.87.
- De tres configuraciones implementadas de *Inception-v3*, el modelo con 156 capas congeladas y 148 capas entrenadas en 50 épocas logra una tasa de acierto de 85.86 %, tasa de error de 14.14 % y una precisión de 0.88. Siendo la arquitectura con los indicadores más altos del entrenamiento en base a *Persea Americana Hass Dataset*.

7. Al aplicar los modelos con mejor desempeño en un conjunto de datos no incluidos en *Persea Americana Hass Dataset*, la arquitectura *Inception-v3* mantiene un mejor desempeño en la identificación de macronutriente en las hojas de Persea Americana variedad Hass. Mientras que la arquitectura *DenseNet169* obtuvo mejores resultados que el modelo *MobileNet*.

8. Se implementó un prototipo para visualizar de manera dinámica los resultados de la identificación del macronutriente que genera anomalías en las hojas de Persea Americana variedad Hass, a través de gráficos circulares y de barras. Con un tiempo de ejecución promedio en la predicción de imágenes aproximado de 2 min 35 s, a consecuencia que los modelos deben ser cargados para en cada proceso de predicción.

Recomendaciones

1. Para trabajos de investigación futuros se recomienda adicionar las categorías de micronutrientes.

2. Se recomienda el uso de la arquitectura *Inception-V3* en trabajos futuros aplicados a otras especies vegetales.

3. Para trabajos futuros se debe considerar emplear otras arquitecturas de redes convolucionales y analizar su desempeño en base al *dataset*.

4. Se recomienda en trabajos futuros el estudio de otras componentes de la Persea Americana (Fruto, tronco, entre otros) que presentan diversos síntomas de deficiencias y enfermedades.

Bibliografía

[git, a] Git.

[git, b] What is GitLab? | GitLab.

[ang, 2019] (2019). Angular - Getting Started with Angular: Your First App. https://angular.io/start.

[boo, 2019] (2019). Bootstrap. https://getbootstrap.com/.

[fla, 2019a] (2019a). Flask 1.0.2 documentation. http://flask.pocoo.org/docs/1.0/.

[fla, 2019b] (2019b). Flask-CORS — Flask-Cors 3.0.7 documentation. https://flask-cors.readthedocs.io/en/latest/.

[fla, 2019c] (2019c). Flask-RESTful — Flask-RESTful 0.3.7 documentation. https://flask-restful.readthedocs.io/en/latest/.

[ker, 2019] (2019). Keras documentation. https://keras.io/.

[mon, 2019a] (2019a). MongoEngine User Documentation — MongoEngine 0.16.3 documentation. http://docs.mongoengine.org/.

[num, 2019] (2019). NumPy — NumPy. https://www.numpy.org/.

[ope, 2019] (2019). OpenCV-Python Tutorials's documentation! — OpenCV-Python Tutorials 1 documentation. https://opencv-python-tutroals.readthedocs.io/en/latest/.

[jup, 2019] (2019). Project Jupyter.

[ten, 2019] (2019). Tensorflow. https://pypi.org/project/tensorflow/.

[pyt, 2019] (2019). Welcome to Python.org.

[mon, 2019b] (2019b). What Is MongoDB?

[Baker and Hearn, 1995] Baker, P. and Hearn, D. (1995). *Gráficas por computadora*. Prentice Hall.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. 35(8):1798–1828.

[Calvo, 2017] Calvo, D. (2017). Red neuronal convolucional CNN.

[Cireşan et al., 2012] Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification.

[Coronado Pérez, 2018] Coronado Pérez, R. R. (2018). Reconocimiento de patrones en imagenes no dermatoscopicas para la detección de enfermedades malignas en la piel, utilizando redes neuronales convolutivas y autocodificadores.

[Coronel Zegarra and Calderón Niquín, 2016] Coronel Zegarra, T. B. and Calderón Niquín, M. A. (2016). Sistema online basado en verificación facial desde dispositivos móviles empleando redes neuronales convolucionales.

[Deng and Yu, 2014] Deng, L. and Yu, D. (2014). Deep learning: methods and applications. 7(3):197–387.

[Erhan et al., 2010] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? page 36.

[Escalera Hueso, 2001] Escalera Hueso, A. d. l. (2001). *Visión por computador: fundamentos y métodos*.

[Foundation, 2019] Foundation, J. (2019). jQuery API Documentation. https://api.jquery.com/.

[Galárraga Cañizares, 2017] Galárraga Cañizares, J. L. (2017). Clasificador de hojas mediante deep learning.

[Goller and Kuchler, 1996] Goller, C. and Kuchler, A. (1996). Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 347–352 vol.1.

[Gonzalez, 2018] Gonzalez, J. L. (2018). Tipos de aprendizaje automático.

[Gonzalez and Woods, 2002] Gonzalez, R. C. and Woods, R. E. (2002). Digital image processing second edition. 455.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[Google, 2019] Google (2019). Google colaboratory. https://colab.research.google.com/notebooks/welcome.ipynb.

[Graves et al., 2009] Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. 31(5):855–868.

[Haykin, 1999] Haykin, S. (1999). Neural networks.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. pages 770–778.

[Howard et al., 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861.

[Huang et al., 2017] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[İrsoy and Cardie, 2014] İrsoy, O. and Cardie, C. (2014). Deep recursive neural networks for compositionality in language. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 27*, pages 2096–2104. Curran Associates, Inc.

[Jaime and Enrique, 2002] Jaime, E. E. J. and Enrique, P. M. L. (2002). Fundamentos de procesamiento de imágenes.

[Le, 2017] Le, J. (2017). The 10 deep learning methods AI practitioners need to apply.

[Lopez Pacheco and Liu, 2017] Lopez Pacheco, M. A. and Liu, W. Y. (2017). Identificación de sistemas no lineales con redes neuronales convolucionales.

[Michalski et al., 2013] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (2013). *Machine Learning: An Artificial Intelligence Approach*. Springer Science & Business Media.

[Mikolov and Zweig, 2012] Mikolov, T. and Zweig, G. (2012). Context dependent recurrent neural network language model. 12(234):8.

[Minagri, 2015] Minagri, M. d. A. y. R. (2015). La palta "producto estrella de exportación".

[Minagri, 2017] Minagri, M. d. A. y. R. (2017). Palta.

[Mitchell, 1997] Mitchell, T. M. (1997). Does machine learning really work? *AI magazine*, 18(3):11–11.

[Montaño Moreno, 2017] Montaño Moreno, J. J. (2017). Redes neuronales artificiales aplicadas al análisis de datos.

[Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press.

[Ortega and Andrés, 2018] Ortega, C. and Andrés, C. (2018). Reconocimiento de marcas de agua embotellada.

[Redolfi et al., 2016] Redolfi, J. A., González Dondo, D., Pucheta, J. A., and Canali, L. R. (2016). Clasificación de variedades de semillas de trigo usando visión por computadora.

[Royal Society (Great Britain), 2017] Royal Society (Great Britain) (2017). *Machine learning: the power and promise of computers that learn by example.* OCLC: 1016323791.

[Ríos et al., 2005] Ríos, D., Corrales, D. M., Daza, G. J., and Aristizábal, A. (2005). *Aguacate: variedades y patrones importantes para Colombia*.

[Sampieri, 2018] Sampieri, R. H. (2018). *Metodología de la investigación: las rutas cuantitativa, cualitativa y mixta*. McGraw Hill Mexico.

[Samuel, 1967] Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. ii—recent progress. *IBM Journal of research and development*, 11(6):601–617.

[Sarle, 1994] Sarle, W. S. (1994). Neural networks and statistical models.

[Sigueñas., 2010] Sigueñas., M. A. E. Y. (2010). *Buenas Prácticas Agrícolas en el Cultivo de Palto*.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[West et al., 2007] West, J., Ventura, D., and Warnick, S. (2007). Spring research presentation: A theoretical foundation for inductive transfer. *Brigham Young University, College of Physical and Mathematical Sciences*, 1:32.

[Whiley et al., 2013] Whiley, A. W., Schaffer, B. A., and Wolstenholme, B. N. (2013). *The Avocado: Botany, Production and Uses*. CABI. Google-Books-ID: r0hpRJca3zEC.

[Xu et al., 2019] Xu, Q., Zhang, M., Gu, Z., and Pan, G. (2019). Overfitting remedy by sparsifying regularization on fully-connected layers of cnns. *Neurocomputing*, 328:69 – 74. Chinese Conference on Computer Vision 2017.

[Yaranga et al., 2018] Yaranga, C. B., Sanchez, Z., and Rodriguez, R. (2018). Transferencia de aprendizaje mediante redes neuronales convolucionales para el reconocimiento de conductores distraidos. *TECNIA*, 28(2).

[Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.