# UNIVERSIDAD PRIVADA ANTENOR ORREGO

## FACULTAD DE INGENIERÍA

PROGRAMA DE ESTUDIO DE INGENIERÍA DE COMPUTACIÓN Y SISTEMAS

TESIS PARA OPTAR EL TÍTULO PROFESIONAL DE

INGENIERO DE COMPUTACIÓN Y SISTEMAS

---

# Detección de Plagas y Enfermedades en los Cultivos de Maíz Utilizando Procesamiento de Imágenes con Redes Neuronales, en el Distrito de Cascas La Libertad, Año 2022

---

Línea de investigación: Sistemas Inteligentes


Autores:

Cipra Salinas, Edder Pier

Rodríguez Alva, Carlos Jhampiere


Jurado evaluador:

Presidente **:** Abanto Cabrera, Heber Gerson

Secretario : Calderón Sedano, José Antonio

Vocal **:** Castañeda Saldaña, José


Asesor:

Gaytán Toledo, Carlos Alberto

Código Orcid: https://orcid.org/0009-0008-8699-7687


Trujillo – Perú

2023


Fecha de Sustentación: 2023/10/06

# Detección de Plagas y Enfermedades en los Cultivos de Maíz Utilizando Procesamiento de Imágenes con Redes Neuronales, en el Distrito de Cascas La Libertad, Año 2022

*por* Carlos Alberto Gaytan Toledo

---

**Fecha de entrega:** 03-oct-2023 09:42a.m. (UTC-0500)
**Identificador de la entrega:** 2184417739
**Nombre del archivo:** Tesis.pdf (3.73M)
**Total de palabras:** 18777
**Total de caracteres:** 110475

Detección de Plagas y Enfermedades en los Cultivos de Maíz Utilizando Procesamiento de Imágenes con Redes Neuronales, en el Distrito de Cascas La Libertad, Año 2022

INFORME DE ORIGINALIDAD

| 2% | 2% | 0% | 0% |
|---|---|---|---|
| INDICE DE SIMILITUD | FUENTES DE INTERNET | PUBLICACIONES | TRABAJOS DEL ESTUDIANTE |

FUENTES PRIMARIAS

| | | |
|---|---|---|
| 1 | andesfiles.s3.sa-east-1.amazonaws.com<br>Fuente de Internet | 2% |

| | | | |
|---|---|---|---|
| Excluir citas | Apagado | Excluir coincidencias | < 2% |
| Excluir bibliografía | Apagado | | |

**Jurado de sustentación Oral**

Ms. Abanto Cabrera Heber Gerson

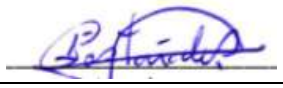
CIP 106421

Presidente

Ms. Calderón Sedano José Antonio

CIP 139198

Secretario



Ms. Castañeda Saldaña José Arturo

CIP 48234

Vocal

Entregado el:

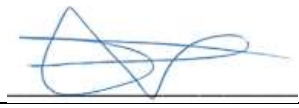
Rodríguez Alva Carlos Jhampiere:

DNI 46043027

Aprobado por:

Cipra Salinas Edder Pier

DNI 76224551



Ms. Gaytan Toledo Carlos Alberto

Asesor de Tesis

# UNIVERSIDAD PRIVADA ANTENOR ORREGO

## FACULTAD DE INGENIERÍA

PROGRAMA DE ESTUDIO DE INGENIERÍA DE COMPUTACIÓN Y SISTEMAS

TESIS PARA OPTAR EL TÍTULO PROFESIONAL DE

INGENIERO DE COMPUTACIÓN Y SISTEMAS

---

# Detección de Plagas y Enfermedades en los Cultivos de Maíz Utilizando Procesamiento de Imágenes con Redes Neuronales, en el Distrito de Cascas La Libertad, Año 2022

---

Línea de investigación**:** Sistemas Inteligentes


Autores:

Cipra Salinas, Edder Pier

Rodríguez Alva, Carlos Jhampiere


Jurado evaluador:

Presidente **:** Abanto Cabrera, Heber Gerson

Secretario **:** Calderón Sedano, José Antonio

Vocal **:** Castañeda Saldaña, José


Asesor:

Gaytán Toledo, Carlos Alberto

Código Orcid: https://orcid.org/0009-0008-8699-7687


Trujillo – Perú
2023


Fecha de Sustentación: 2023/10/06

## DECLARACION DE ORIGINALIDAD

Yo, Gaytán Toledo Carlos Alberto, Docente del Programa de Estudio de pregrado del Programa de Estudio de Ingeniería de Computación y Sistemas de la Universidad Privada Antenor Orrego, asesor de la tesis titulada “Detección de Plagas y Enfermedades en los Cultivos de Maíz Utilizando Procesamiento de Imágenes con Redes Neuronales, en el Distrito de Cascas La Libertad, Año 2022”, de los autores Rodríguez Alva Carlos Jhampiere y Cipra Salinas Edder Pier, dejo constancia de lo siguiente:

• El mencionado documento tiene un índice de puntuación de similitud del 2%. Así lo consigna el reporte de similitud emitido por el software Turnitin el día 03 de Octubre del 2023.

• He revisado con detalle dicho reporte de la tesis “Detección de Plagas y Enfermedades en los Cultivos de Maíz Utilizando Procesamiento de Imágenes con Redes Neuronales, en el Distrito de Cascas La Libertad, Año 2022.” y no se advierte indicios de plagio.

• Las citas a otros autores y sus respectivas referencias cumplen con las normas establecidas

por la Universidad.

Trujillo, 03 de Octubre de 2023


Gaytán Toledo Carlos Alberto

Dni: 18112150

ORCID:

https://orcid.org/0009-0008-8699-7687

Rodríguez Alva Carlos Jhampiere

Dni: 46043027

Cipra Salinas Edder Pier

Dni: 76224551

## Dedicatoria

A mis padres Gloria y Bernabe; por su esfuerzo y sacrificio para lograr que sus hijos seamos unos excelentes profesionales; y motivación para seguir luchando por nuestros sueños.

A mi esposa e hija por ser el motivo para cumplir cada una de mis metas trazadas.

A mis familiares por su apoyo incondicional y ser una inspiración para ellos.

A mis compañeros de la Universidad y docentes que siempre me brindaron su apoyo e inculcaron valores y principios durante mi formación Profesional.

***Edder Pier Cipra Salinas***

A María Julia Alva León y Hermógenes Vílchez Coronado por todo el apoyo que me brindaron, por sus consejos y buenos deseos.

A familiares y amigos. A todos los Docentes por ser el soporte para adquirir nuevos conocimientos y ponerlos en práctica. Gracias.

***Carlos Jhampiere Rodríguez Alva***

## Agradecimientos

A Dios, por guiarnos y mantenernos firmes desde los inicios de nuestra carrera Profesional como Ingeniero de Computación y Sistemas.
A nuestro asesor, Ing. Gaytán Toledo Carlos Alberto por su apoyo y orientación en el proceso de Trabajo de Tesis.
A los agricultores del Distrito de Cascas, quienes nos proporcionaron las facilidades e información para el desarrollo del presente Trabajo de Tesis.

Gracias

**Los Autores**

## Resumen

**“DETECCIÓN DE PLAGAS Y ENFERMEDADES EN LOS CULTIVOS DE MAÍZ UTILIZANDO PROCESAMIENTO DE IMÁGENES CON REDES NEURONALES, EN EL DISTRITO DE CASCAS – LA LIBERTAD, AÑO 2022"**

Por:

Br. Carlos Jhampiere Rodríguez Alva

Br. Edder Pier Cipra Salinas

Los agricultores en el Distrito de Cascas no cuentan con un procedimiento que les ayude a prevenir o a detectar a tiempo el tipo de plaga o enfermedad que ataca sus cultivos de maíz.

La presente tesis tiene como objetivo principal, entrenar y evaluar 4 modelos con redes neuronales como ResNet50. EfficientNetVB0, MobileNetV2 y DenseNet201. Para ello se recopilaron un conjunto de imágenes, las que se clasificaron en; imágenes de Tizón de la Hoja de Maíz (*Blight*), Roya Común (*Common Rust*), Gusano (Armyworm), y plantas con Hojas sanas (*Healthy*).

Luego se procedió agruparlos una vez más como imágenes de entrenamiento, validación y prueba. Para el desarrollo se aplicaron técnicas de aumentación, filtros, funciones. Con el objetivo de mejorar el aprendizaje de los modelos de redes neuronales. Luego se procede a medir que tan precisos son los modelos.

Se realiza pruebas con hojas de Maíz con Gusano, Roya Común, Tizón de la Hoja de Maíz y sanas. Donde se obtuvo como resultado que el mejor modelo en términos de rendimiento es el DenseNet201. En el caso de la Plaga Gusano tiene una precisión de 0.97 que pertenece a esa categoría. Para Roya Común con 0.95 de precisión. Tizón de la Hoja al 0.98 de precisión. Saludable 0.73 de precisión. Recalcando que son imágenes desconocidas para los modelos.

Para comprobar la certeza de los modelos se utilizó la matriz de confusión que mide que tan acertados son sus predicciones de cada modelo. Dando a MobileNetV2 como el modelo que tiene un mejor acierto con respecto a los demás modelos con un 99% de certeza. Mientras el entrenamiento tenga un gran número de imágenes, filtros, técnicas y funciones adecuadas. Un modelo puede llegar a dar resultados que ayuden a tomar mejores decisiones por sus niveles de precisión que pueden llegar a obtener.

**Palabras Clave:** Convolución, Sobre Ajuste, Dropout, Función de Activación, Generalización.

## Abstract

**"DETECTION OF PESTS AND DISEASES IN CORN CROPS USING IMAGE PROCESSING WITH NEURAL NETWORKS, IN THE DISTRICT OF CASCAS - LA LIBERTAD, YEAR 2022"**

**By:**

Br. Carlos Jhampiere Rodríguez Alva

Br. Edder Pier Cipra Salinas

Farmers in the Cascas District do not have a procedure to help them prevent or detect in time the type of pest or disease that attacks their corn crops.

The main objective of this thesis is to train and evaluate 4 models with neural networks such as ResNet50. EfficientNetVB0, MobileNetV2 and DenseNet201. For this purpose, a set of images were collected and classified into; images of Corn Leaf Blight (Blight), Common Rust (Common Rust), Armyworm (Armyworm), and plants with Healthy Leaves (Healthy).

They were then grouped once again as training, validation and test images. For the development, augmentation techniques, filters and functions were applied. In order to improve the learning of neural network models. Then we proceed to measure how accurate the models are.

Tests were carried out on corn leaves with worm, common rust, corn leaf blight and healthy leaves. The result was that the best model in terms of performance is DenseNet201. In the case of Pest Worm it has an accuracy of 0.97 which belongs to that category. For Common Rust with 0.95 accuracy. Leaf Blight at 0.98 accuracy. Healthy 0.73 accuracy. Emphasizing that these are unknown images for the models.

To test the accuracy of the models we used the confusion matrix which measures how accurate their predictions are for each model. Giving MobileNetV2 as the model that has a better accuracy with respect to the other models with 99% certainty. As long as the training has a large number of images, filters, techniques and adequate functions. A model can give results that help to make better decisions because of the levels of accuracy that can be obtained.

**Keywords:** Convolution, augmentation, Relu, Dense, Perceptron, Overfitting, Dropout, Bias, Activation Function, Normalization, Gradient Descent.

## Presentación

Señores miembros del Jurado:

De conformidad con lo estipulado en el Reglamento de Grados y Títulos de la Universidad Privada Antenor Orrego, ponemos a su disposición el informe de tesis titulado “Detección de Plagas y Enfermedades en los Cultivos de Maíz Utilizando Procesamiento de Imágenes con Redes Neuronales, en el Distrito de Cascas La Libertad, Año 2022” para que sea revisado y evaluado y de ser aprobado pueda ser defendido oralmente para optar el título profesional de Ingeniero de Computación y Sistemas.

De antemano, nos excusamos de los errores involuntarios en que se hubiera incurrido en el desarrollo y redacción del misma, esperando del honorable jurado un justo dictamen.

Apellidos y Nombres: Rodríguez Alva Carlos Jhampiere

Apellidos y Nombres: Cipra Salinas Edder Pier

## Tabla Contenido

# Índice de Tablas

# Índice de Figuras

# I. INTRODUCCION

## 1.1. Problema de Investigación

Las imágenes suelen ser utilizadas en diferentes escenarios para llevar a cabo una tarea o proceso. Tal es el caso de la visión por computador que utiliza imágenes para identificar un objeto. El avance tecnológico ha desarrollado redes neuronales artificiales inspiradas en el cerebro humano para llevar a cabo el procesamiento de imágenes y obtener de ellas información relevante en la medicina, agricultura, seguridad ciudadana, política, tecnología, etc. Esta información extraída puede ser útil para descubrir patrones, objetos, predecir y tomar decisiones ante una situación difícil. (Trask, 2019)

El cultivo del maíz es el principal cereal en el mundo, lo cual representa uno de los aportes más valiosos a la seguridad alimentaria a nivel mundial, junto al arroz y el trigo son considerados los 3 cereales más cultivados en el mundo. Siendo EE. UU el principal productor de maíz, seguido de China y Brasil, quienes con alrededor del 47% de la superficie mundial sembrada con maíz (184 M ha). (Mendoza, 2017)

Según el Gobierno Mexicano las plagas más comunes se clasifican según las partes que ataque en la planta:

| Parte atacada | Nombre de la Plaga |
|---|---|
| **Rizófagas**<br><br>**(Atacan raíz)** | ❖ Gallina ciega<br>❖ Gusano de alambre<br>❖ Diabrótica |
| **Plagas del Follaje** | ❖ Gusano Cogollero<br>❖ Gusano Elotero<br>❖ Gusano Soldado<br>❖ Araña Roja |

*Tabla 1.* *Nombres Comunes de Plagas*

| Planta con Plaga | Planta con Enfermedad |
|---|---|

Figura 1. *Plagas y Enfermedades que atacan al cultivo de Maíz*

Por ello si no se combate bien una plaga o enfermedad. Las consecuencias podrían generar daños de grandes proporciones. Los brotes muchas veces dependen de diversos factores como; el patógeno, el manejo del cultivo y el medio ambiente. (México, 2019)

Según la Organización de las Naciones Unidas para la Alimentación y la Agricultura (FAO), estima que las plagas destruyen cada año hasta un 40 % de la producción global de cultivos, mientras que las enfermedades que padecen los cultivos cuestan anualmente a la economía mundial más de 220.000 millones de dólares. La principal causa se debe sobre todo al cambio climático. (ONU, 2021)

El Perú posee entre 60 y 70% de biodiversidad; pero se ve afectada por factores como contaminación y deforestación, los mismos que degradan el ecosistema de diversas especies ya sean animales o plantas. Las ultimas que son afectadas por contaminación de las cuencas hidrográficas. Esto se debe a sistemas de extracción o técnicas no controladas. Ocasionando el desequilibrio en los recursos naturales. Y como consecuencia daña los sembríos de la Agricultura Peruana. (Riego, 2018)

Se suman otras causas que afectan directa o indirectamente el rendimiento agrario; que están relacionadas a nivel económico, social, ambiental e

institucional. La innovación tecnológica es uno los pocos o escasos recursos que se aplican hoy en día en el Perú. Es decir, aspectos olvidados o de poca importancia para las autoridades. (Agricultura, 2016)

Uno de los departamentos con mayor producción Agrícola en el Perú es el departamento de la Libertad. El mismo que se ubica como el cuarto productor de maíz, con 138,1 mil toneladas. En conjunto que es uno de los mayores consumidores de este producto, no solo porque el maíz es un insumo para la producción de otro producto final; sino existe gran demanda avícola. Lo que la hace una de las regiones con gran consumo del mencionado cereal. (Sineace, 2020)

Una de las preocupaciones que a diario se enfrentan los agricultores son las plagas y enfermedades que afectan el rendimiento de sus cultivos; existe un organismo denominado SENASA, el mismo que se encarga de llevar el control de plagas y enfermedades de algunos cultivos en la Región la Libertad; por lo que ellos aplican técnicas poco precisas. Considerando como parámetro de evaluación: una Hoja, fruto, planta, raíz, implementación de trampas; lo que les permite determinar un diagnóstico. (Senasa, 2017)

Las plagas o enfermedades se pueden presentar en las diferentes fases de crecimiento de la planta.

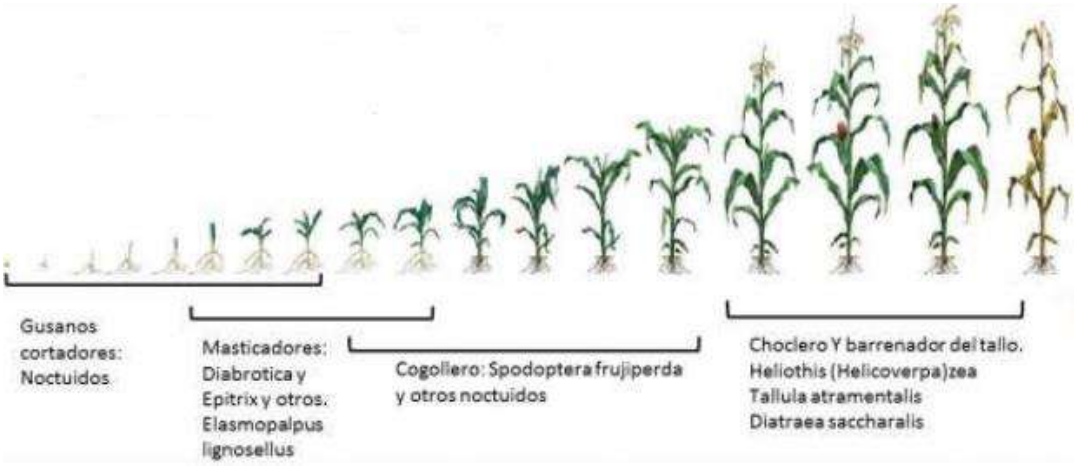



Figura 2. *Plagas y Enfermedades que atacan durante el crecimiento del Maíz*

Parámetros de control para el monitoreo de plagas y enfermedades a considerar son los siguientes:

- ✓ Tipo de insecto o enfermedad
- ✓ Estado fenológico
- ✓ Unidad de muestreo (total de plantas por área). (Bazán, 2016)

A. Delimitación del Problema

La Tesis en mención se delimita en la evaluación y comparación de resultados de los modelos seleccionados de Redes Neuronales, utilizando técnicas de procesamiento de imágenes para detección de plagas y enfermedades en los cultivos de maíz, con la finalidad de brindar un aporte tecnológico a los agricultores en el Distrito de Cascas, La Libertad, durante el año 2022.

B. Características Problemáticas

Durante las diferentes fases del crecimiento del maíz, los agricultores del Distrito de cascas presentan los siguientes problemas:

- ✓ Plagas en los cultivos del maíz (Gusano cogollero)
- ✓ Enfermedades que atacan a los cultivos del maíz (Tizón de la hoja de maíz y roya común)
- ✓ Desconocimiento del uso de herramientas Tecnológicas
- ✓ Aplicación de insecticidas sin tener la certeza del tipo de plaga o enfermedad que ataca su cultivo.
- ✓ Cultivos con deficiencia en desarrollo.
- ✓ Pérdidas de inversión.

C. Definición del Problema

En el Distrito de Cascas, La Libertad, los agricultores enfrentan desafíos en la detección temprana de plagas y enfermedades en los cultivos de maíz. Esta falta de detección oportuna puede resultar en pérdidas significativas de cultivos y afectar negativamente en la producción agrícola y la economía local.

D. Formulación del Problema

¿Cómo mejorar la detección temprana de plagas y enfermedades en los cultivos de maíz en el Distrito de Cascas, La Libertad?

## 1.2. Objetivos

### 1.1.1. Objetivo General

Identificar el modelo más eficaz de procesamiento de imágenes con Redes Neuronales para la detección de plagas y enfermedades en los cultivos de maíz en el Distrito de Cascas, La Libertad, durante el año 2022.

### 1.1.2. Objetivos Específicos

- Recopilar imágenes de plagas y enfermedades de los cultivos de maíz.
- Entrenar los modelos de redes neuronales utilizando técnicas de procesamiento de imágenes para la detección de plagas y enfermedades en los cultivos de maíz.
- Analizar datos relevantes sobre plagas y enfermedades comunes en los cultivos de maíz en el Distrito de Cascas. Utilizando técnicas de Procesamiento de imágenes.
- Evaluar los modelos para la detección de plagas y enfermedades en los cultivos de maíz. Para determinar su precisión y rendimiento de detección en plagas y enfermedades.
- Visualizar los resultados de la predicción de cada modelo.
- Interpretar los resultados obtenidos.
- Identificar el mejor modelo para la detección de plagas y enfermedades en los cultivos de maíz.

## 1.3. Justificación del Estudio

### A. Importancia de la Investigación

La importancia de la investigación se centra en varios aspectos que benefician a los agricultores del Distrito de Cascas, La Libertad. A continuación, se detallan algunos puntos clave que destacan la relevancia de este estudio:

- *Mejora en la Producción Agrícola.*

  La detección temprana y precisa de plagas y enfermedades en los cultivos de maíz permite a los agricultores tomar medidas preventivas y aplicar tratamientos específicos de manera oportuna. Esto ayudará a reducir las pérdidas de cultivo y mejorar la producción agrícola en el Distrito de Cascas.

- *Eficiencia en el Uso de Recursos.*

  Al utilizar un modelo de Redes Neuronales para la detección de plagas y enfermedades en imágenes de cultivos, reduce la necesidad de inspecciones manuales extensas y costosas. Esto implica un uso más eficiente de recursos como el tiempo y el personal agrícola, lo que a su vez puede generar ahorros y optimizar los recursos disponibles.

- *Seguridad Alimentaria.*

  El maíz es un cultivo básico en la alimentación humana y animal. Al mejorar la detección y control de plagas y enfermedades en los cultivos de maíz, se contribuye a garantizar una oferta estable de alimentos y a fortalecer la seguridad alimentaria en el Distrito de Cascas.

- *Apoyo a la Toma de Decisiones.*

  El modelo de detección de plagas y enfermedades proporcionará a los agricultores información valiosa y precisa sobre el estado de salud de sus cultivos. Esto permite toma de decisiones más informada en cuanto a la

aplicación de tratamientos específicos y otras medidas de control, evitando el uso excesivo o inadecuado de agroquímicos y promoviendo prácticas agrícolas más sostenibles.

- *Contribución al Conocimiento Científico.*

  La investigación en el campo de la detección de plagas y enfermedades mediante Redes Neuronales en cultivos de maíz aportará nuevos conocimientos y perspectivas a la comunidad científica y académica. Los resultados y las lecciones aprendidas podrán ser utilizados como base para futuras investigaciones y aplicaciones en otros cultivos y regiones.

- *Aplicabilidad a otras regiones.*

  El modelo de detección de plagas y enfermedades y sus resultados obtenidos en la investigación podrían ser transferible y aplicable a otros cultivos o regiones con problemáticas similares.

B. Viabilidad de la Investigación

- *Acceso a Tecnología y Recursos:*

  El procesamiento de imágenes con Redes Neuronales requiere de tecnología computacional adecuada y recursos de hardware suficientes para el entrenamiento y evaluación del modelo.

  Para ello se pueden utilizar servicios en la nube que ofrecen capacidad de procesamiento y almacenamiento escalable para cumplir con los requerimientos de la investigación, servicios o herramientas como Google Colab(Gpu, Tpu), Google Drive, Tensorflow, Keras, etc. Herramientas y Plataformas de Aprendizaje Profundo disponibles que facilitan el desarrollo de este tipo de proyectos.

- *Disponibilidad de Datos:*

  La investigación se basó en la toma de imágenes de campo y del repositorio Kaggle de cultivos de maíz infectados con plagas,

enfermedades y sanas. Y referente a los modelos se hace uso de modelos pre entrenados con el repositorio IMAGENET.

- *Conocimiento Técnico:*
  Conocimientos previos en lenguajes de programación como Python, Librerías (keras, tensorflow, Matplotlib) y servicios en la nube que fueron utilizados para dicho propósito de la Investigación.

- *Colaboración Local:*
  Los agricultores del Distrito de Cascas brindaron el acceso a sus parcelas para recolección de imágenes en sus diferentes etapas de crecimiento del maíz e información relevante para el desarrollo de nuestra Tesis.

C. Aportes

- *Avance en la Tecnología Agrícola:*
  La investigación propone una aplicación innovadora de la tecnología de Redes Neuronales en la detección de plagas y enfermedades en cultivos de maíz. Este enfoque representa un avance importante en la aplicación de técnicas de inteligencia artificial en la agricultura, contribuyendo a la modernización y optimización de las prácticas agrícolas.

- *Herramienta de Toma de Decisiones:*
  El modelo de detección desarrollado proporciona a los agricultores una herramienta precisa y confiable para tomar decisiones informadas sobre la salud de sus cultivos. Este aporte es especialmente valioso en un contexto en el que la detección temprana es crucial para prevenir pérdidas y asegurar una producción exitosa.

- *Reducción de Pérdidas Agrícolas:*

  Uno de los principales aportes de tu investigación es la contribución a la reducción de pérdidas en la producción agrícola. La detección temprana y precisa de problemas fitosanitarios permite a los agricultores aplicar medidas de control oportunas, lo que se traduce en una disminución de las pérdidas de cultivos y un aumento en la productividad.

- *Promoción de la Agricultura Sostenible:*

  Al utilizar la tecnología como medio para obtener datos mas precisos al momento de determinar que procedimiento se debe aplicar para curar o mitigar el daño de las plagas o enfermedades que ocasionan en los cultivos. La investigación contribuye a una agricultura más sostenible. Esto implica una reducción en el uso de agroquímicos y, por ende, un menor impacto ambiental, preservando los recursos naturales y la biodiversidad.

- *Transferencia de Conocimiento:*

  Los resultados y las lecciones aprendidas de tu investigación pueden ser compartidos con la comunidad agrícola, instituciones gubernamentales y la comunidad científica en general. Este intercambio de conocimiento puede fomentar la adopción de prácticas más eficientes y tecnológicas en el sector agrícola, impulsando el desarrollo agrícola a nivel local y nacional.

## II. MARCO DE REFERENCIA

### 2.1. Antecedentes del Estudio

(Agarwal, y otros, 2019)“A convolution Neural Network based approach to detect the disease in Corn Crop” El objetivo del Proyecto es identificar enfermedades en las hojas de Maíz, donde se aplicó un modelo basado en CNN con 3 capas de convolución, 3 de Maxpooling y 2 capas totalmente conectadas. Para ello se comparó los resultados de rendimiento con otros modelos, considerando varias métricas. Se determinó que en términos de precisión el modelo VGG16 tiene un mejor rendimiento sobre 1%. Por motivo de que su estructura es mucho más grande que el modelo propuesto. En cambio, con respecto al tiempo de procesamiento el modelo propuesto es más eficiente.

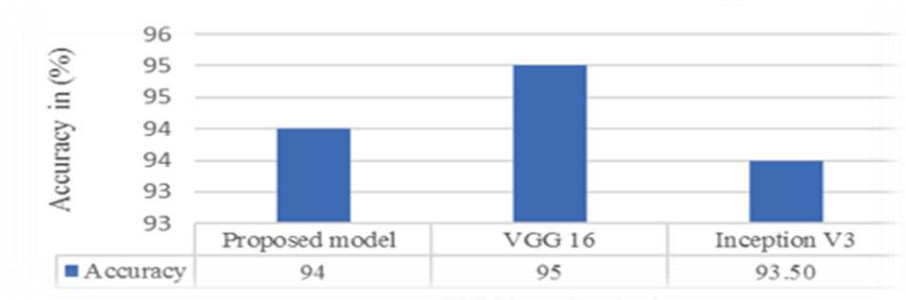



Figura 3. *Comparativo de Modelos CNN.*

Comparación de los modelos pre entrenados y del modelo propuesto en términos de precisión.

**Nota:** El gráfico representa la Comparación del rendimiento de los modelos pre entrenados y del modelo propuesto en términos de precisión. *Mohit* Agarwal, Vijay Kumar Bohat, 2019. IEEE

Se concluyó que el modelo propuesto tiene una precisión de 94% en comparación a los demás.

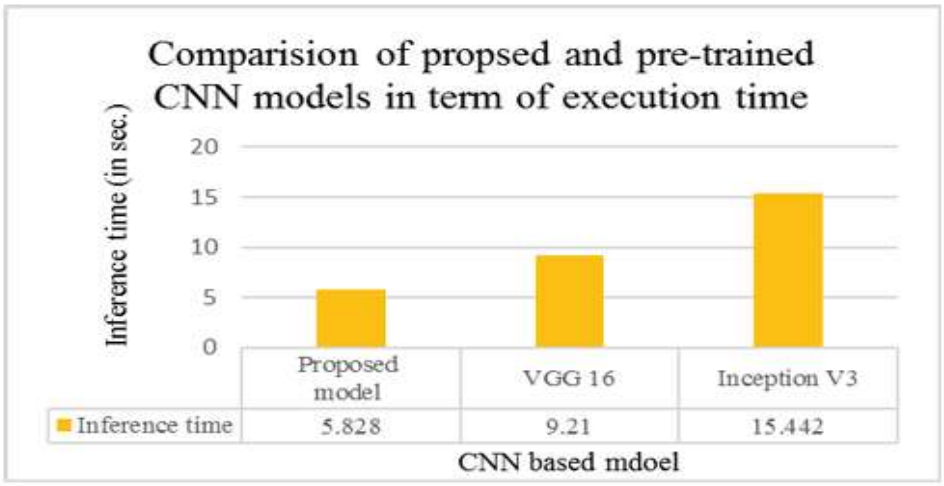



Figura 4. *Eficiencia de Modelos en términos de tiempo para predecir*

**Nota:** El gráfico representa la Comparación del rendimiento de los modelos pre entrenados y del modelo propuesto en términos de tiempo de inferencia. Mohit Agarwal, Vijay Kumar Bohat, 2019. IEEE

(Mehmet Metin Ozguven, 2019)“Automatic detection and classification of leaf spot disease in sugar beet using deep learning algorithms” El Proyecto propuesto tiene como finalidad detectar y clasificar la enfermedad que ataca a la plantación de remolacha azucarera mediante algoritmo de Deep Learning. Por lo cual se usó dos modelos el Faster R-CNN y Update Faster CNN. Donde al modelo CNN se le realizó unos ajustes en sus parámetros, de su arquitectura. Dando un resultado del 95.48% en la detección y clasificación de enfermedades. Por ello se concluye que la actualización de parámetros fue favorable para la detección de la enfermedad en comparación a otros modelos.

Comparación de las tasas de éxito de los métodos propuestos en la detección de enfermedades en las plantas.

| Número de imágenes | Métodos | Sens. | Espec. | TCC |
|---|---|---|---|---|
| 155 | R-CNN más rápido | 92.89 | 92.89 | 92.89 |
| | R-CNN actualizado y más rápido | 95.48 | 95.48 | 95.48 |

*Tabla 2. Tabla de éxitos de modelos propuestos*

**Nota:** La tabla representa el éxito obtenido de los modelos propuestos. Mehmet Metin Ozguven, Kemal Adem, 2019. ELSEVIER.

(Yanfen Li, 2020)"Crop pest recognition in natural scenes using convolutional neural networks" En la presente investigación se propuso 5 modelos de Deep Learning de los cuales se seleccionó el más eficiente. Para clasificar diez tipos de plagas en los cultivos. Para ello se recopilo las imágenes y se aplicó un método de aumento de datos para obtener un mayor número de imágenes con las técnicas de rotación, reflejo, adición de ruido y zoom. Luego en el proceso de pre procesamiento se usó técnicas como Umbral, Detección de Contornos y Algoritmo de Cuenca Hidrográfica. Seguido de las técnicas mencionadas se aplicó el Algoritmo GrabCut para eliminar el fondo de la imagen.

En la comparación de resultados se determinó que el modelo GoogleNet tiene una mejor precisión con respecto a los demás modelos.
Comparación de los modelos propuestos en términos de Precisión.

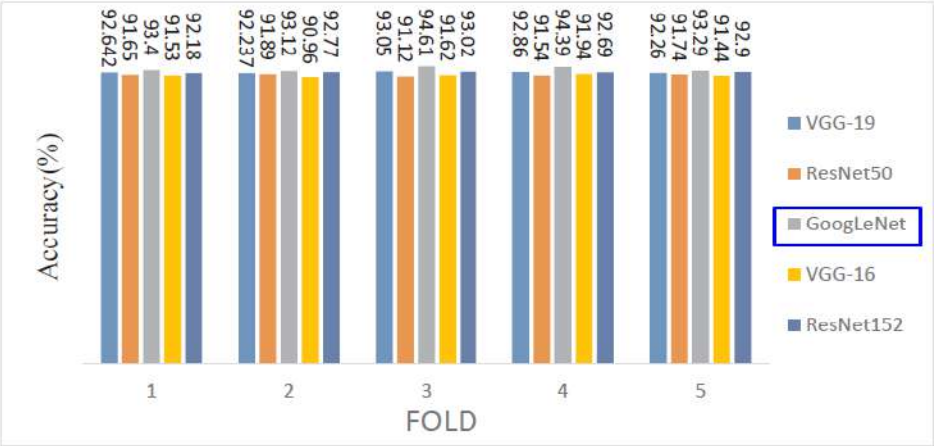



Figura 5. *Modelos con niveles de precisión regular*

**Nota:** El gráfico representa el nivel de precisión de cada modelo. Determinado que GoogleNet tiene mejor precisión. Yanfen Li, 2020. ELSEVIER.

Luego los resultados en la Clasificación de las diez plagas con el modelo GoogleNet tiene una precisión del 98,91%.

(Salvador Gutiérrez, 2021)“Deep learning for the differentiation of downy mildew and spider mite in grapevine under field conditions”. En el Proyecto propuesto su principal objetivo es identificar y diferenciar la enfermedad Mildiú de la plaga araña roja en el cultivo de vid utilizando modelos de Deep Learning. El proceso se dividió en 3 etapas. Etapa 1 adquisición de imágenes (Hoja con mildiú, hoja con araña roja y hoja sana) con un total de 250 imágenes de cada clase (enferma, con plaga y sana). En la Etapa 2 se realizó el pre procesamiento de imágenes con el recorte del área de interés, se aplicó el algoritmo de segmentación denominado GrabCut para lograr la eliminación del fondo. Luego por temas de iluminación invariable en las imágenes y que se requería para tener una mayor precisión; era usar el modelo HSV en reemplazo del RGB. Para así tener uniformidad en los colores de iluminación o intensidad sin afectar el tono de las imágenes.

Etapa 3 se aplicó el entrenamiento del modelo CNN con la estructura que se muestra a continuación:

Arquitectura de una Red Neuronal Convolucional, usada para el entrenamiento de un modelo de clasificación.

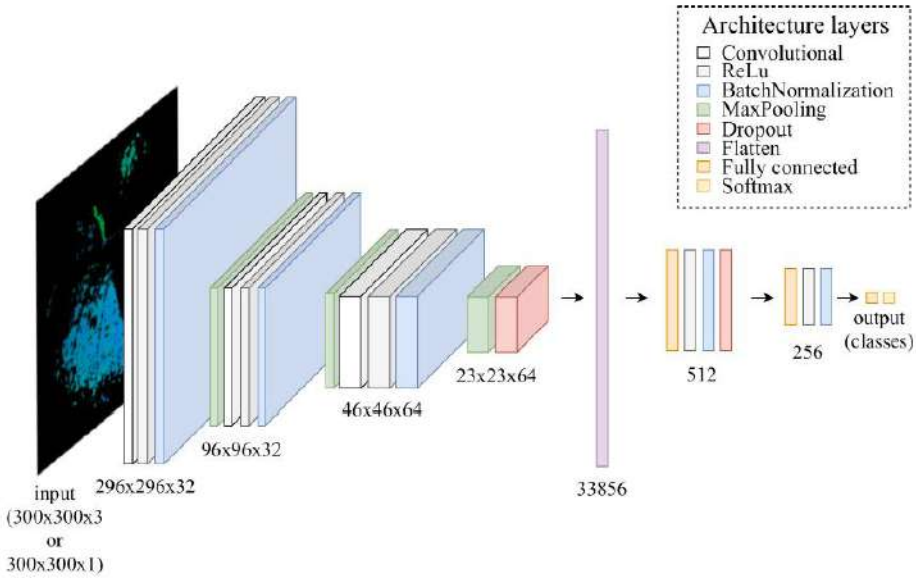



Figura 6. *Arquitectura de una CNN*

**Nota:** El gráfico representa la arquitectura de una CNN, con los procesos de identificación de objeto. Salvador Gutiérrez, 2021. ELSEVIER.

Para ello se utilizó dos esquemas uno con HSV y el otro solo con H(tono), el cual se consideró el más resaltante para la detección. Obteniendo los siguientes resultados: con una precisión en la fase de entrenamiento 0.94, fase de validación 0.93, y fase de pruebas 0.94. Usando HSV entrenamiento 0.90, validación 0.92 y pruebas 0.86. Solo utilizando el esquema H entrenamiento 0.95, validación 0.90 y pruebas 0.87.

(Ramar Ahila Priyadharshini, 2019), “Maize leaf disease classification using deep convolutional neural networks”, la presente investigación tiene como objetivo clasificar enfermedades de la hoja de maíz y evaluar el nivel de

eficiencia de la arquitectura LeNet. Realizando variaciones en sus parámetros como el tamaño y la profundidad de las capas convolucionales. Para ello se aplicó el algoritmo de descenso de gradiente que se usa para entrenar el modelo. Las imágenes usadas para entrenamiento se obtuvieron del repositorio de PlantVillage. Donde se obtuvo un conjunto de cuatro clases diferentes: imágenes de hoja sana, y las otras tres clases se conforman de las enfermedades comunes.

El proceso que se llevó a cabo fue: selección de imágenes, preprocesamiento; donde se utilizó PCA (Análisis de los principales Componentes), que consiste en estandarizar las dimensiones de las imágenes con el fin de mejorar el entrenamiento para obtener mejores resultados. Luego se procede al entrenamiento del Modelo LeNet. Un aspecto importante para el modelo es; su arquitectura debido a que cuenta con cinco capas; dos capas convolucionales y tres capas completamente conectadas. Como resultado final de precisión es de 97.89%. lo cual se determina que es un modelo eficiente para la clasificación de enfermedades.

Comparación de resultados en términos de precisión con Métodos propuestos.

| Method | Classification Accuracy % |
|---|---|
| GA-SVM | 92.82 |
| Artificial Neural Network Classifier | 94.4 |
| Support Vector Machine | 89.6 |
| Our Proposed Methodology | **97.89** |

*Tabla 3. Precisión en modelos propuestos*

**Nota:** La tabla representa el nivel de precisión de los métodos de clasificación. Ramar Ahila Priyadharshini, 2019. ELSEVIER.

## 2.2. Marco Teórico

(Jinzhu Lu, 2021), “Review on Convolutional Neural Network (CNN) Applied to Plant Leaf Disease Classification”. Deep Learning basado en algoritmos de redes neuronales (De mejor rendimiento perceptrón multicapa, cnn, y red neuronal recurrente), usados para la extracción de características de datos.

Aquí se describe como trabaja una CNN que consta de capas convolucionales, de agrupación y totalmente conectadas.

Este proceso inicia en la capa de convolucional que usa la información referencial de las imágenes para extraer las características. Empezando en la parte superior de la imagen de izquierda a derecha, va tomando los valores de cada píxel en bloques(kernel), multiplicando los valores, seguido se suma y se agrega 1 sesgo al final de cada kernel. Y así procesa cada pixel repitiendo el mismo mecanismo hasta lograr filtrar las características mas importantes de una imagen.

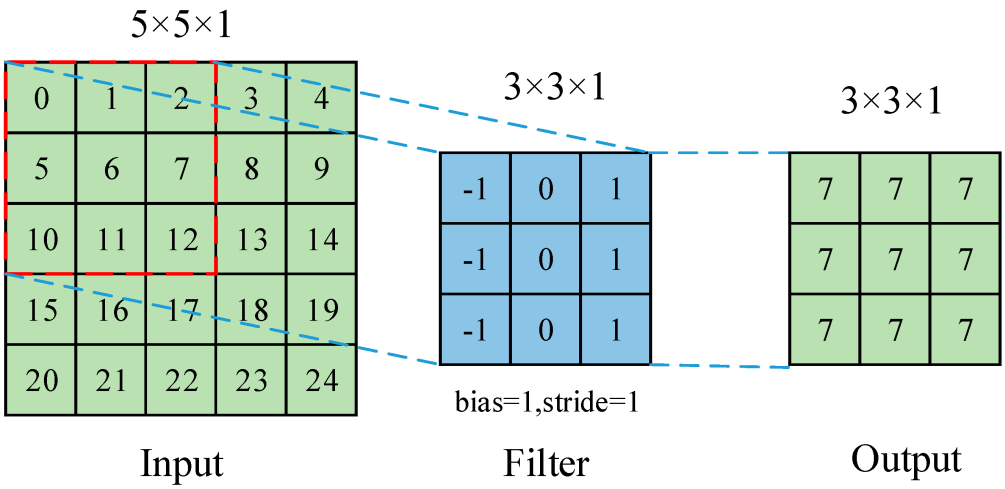



Figura 7. *Proceso de extracción de características en la convolución*

La capa de agrupación procede seleccionando los pixeles del mapa de características de la capa superior mediante el muestreo y evitando variación en la traslación, rotación y escalado del modelo de cnn. Las técnicas de agrupación más usadas son la máxima o media. La máxima consiste en

dividir la imagen de entrada en varias regiones rectangulares según el filtro y generar el valor máximo para cada región.

Para la agrupación promedio, su salida es el promedio de cada región.

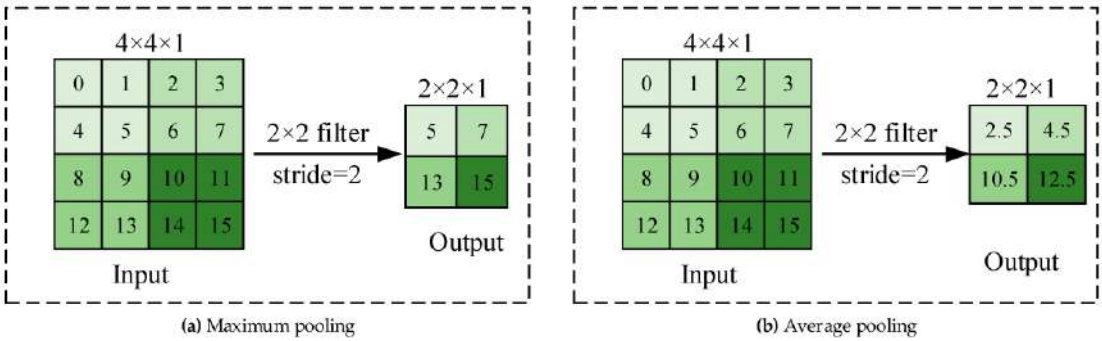



Figura 8. *Procesos de Agrupación más usados.*

Los algoritmos o modelos cnn más usados para tareas de clasificación se conocen a: AlexNet, VGGNet, GoogleNet, ResNet, MobileNet y EfficientNet.

A continuación, se visualiza una la arquitectura de un modelo CNN utilizado para determinar las enfermedades de las hojas de calabaza. Donde se ingresa un dataset de imágenes para luego realizar el proceso de aplanamiento del mapa de características en el clasificador. Por ello se describe en 3 pasos:

Paso 1: Preparación de los datos y preprocesamiento.

Donde las imágenes fueron divididas en 3 conjuntos entrenamiento para el que el modelo aprenda, validación para configurar los hiperparámetros durante el entrenamiento y prueba que son datos desconocidos para el modelo y de los cuales determinará cual es la enfermedad que está afectando a la hoja de calabaza. También se aplican técnicas de limpieza de datos.

Paso 2. Construcción, entrenamiento y evaluación del Modelo

Cuando se ha construido el modelo, se definen varios hiperparámetros que van a mejorar el proceso de entrenamiento y evaluación. Los autores

mencionan que usar combinaciones de parámetros y utilizar un método de búsqueda para la iteración, ayuda a mejorar la eficiencia del modelo. El proceso es que al entrenar una cnn los datos se ubican en la primera capa y cada neurona va actualizando el peso de la neurona a través de un proceso denominado propagación hacia atrás. Según si la salida es igual a la etiqueta ingresada. Este proceso se repite hasta encontrar el mejor

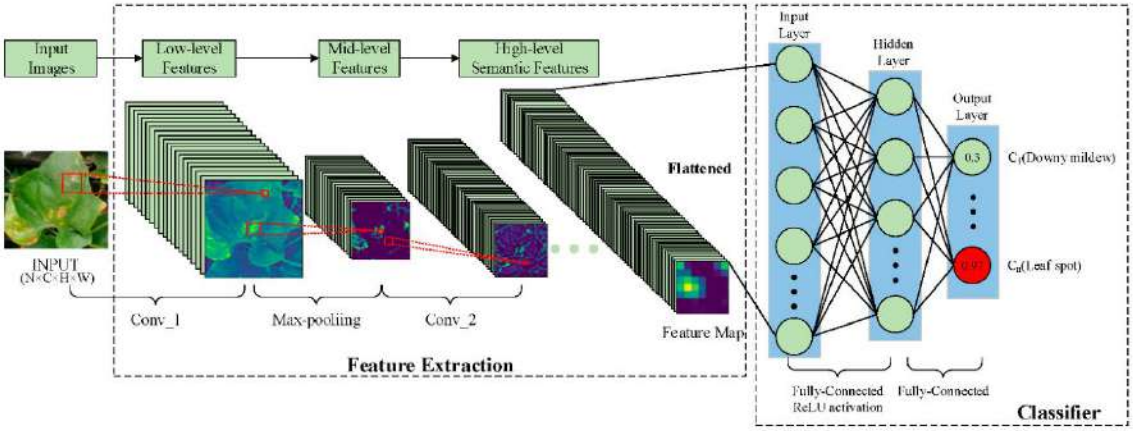



Figura 9. *Arquitectura de CNN*

Se menciona una de las herramientas para evolución del rendimiento de un modelo de cnn. Con el objetivo de medir que tan bueno es un modelo de red neuronal. Esa herramienta es la Matriz de confusión la misma que mostrará los resultados acertados e incorrectos. Por ende, se compone de 4 variables. TP verdadero positivo, FP falso positivo, TN verdadero negativo y FN falso negativo.

Paso 3. Inferencia e implementación.

Lo primero es que tan bueno es el modelo para predecir un resultado y mostrar el resultado adecuado. La implementación se refiere al implantar el modelo a un dispositivo, la nube o cualquier otra tecnología que se utilice para estos fines.

(John D. Kelleher, 2015) “Fundamentals of Machine Learning for Predictive Data Analytics” En el presente libro se menciona un recurso para evaluar el rendimiento de un modelo de cnn, se define como una herramienta de análisis. Específicamente cuando se realiza predicciones mediante modelos neuronales, necesitan ser evaluados para saber que tan buenos son los modelos aplicados en determinada situación. Para ello el procesamiento de datos binarios tiene dos niveles positivo y negativo. Los mismo que se distribuyen en 4 resultados cuando un modelo predice y que son aprovechados por una matriz de confusión:

TP: Verdaderos positivos: valor real es positivo y el predicho también es positivo.

TN: verdaderos negativos: valor real es negativo y el predicho también es negativo.

FP: falsos positivos:  valor real es negativo y el predicho es positivo.

FN: falsos negativos: valor real es positivo y el predicho es negativo.

De ello se resume que existe 2 predicciones correctas que son TP (positiva verdadera) – TN (negativa verdadera), y 2 predicciones incorrectas que son FP (falso positivo) – FN (false negativo).

A continuación, se muestra la estructura de la matriz de confusión distribuido, en las columnas representa los valores de las predicciones realizadas por el modelo. Mientras que en las filas son los valores reales (imágenes o datos sin procesar) que representarían los valores esperados de la característica objetivo que se quiere obtener.

| | | Prediction | |
|---|---|---|---|
| | | positive | negative |
| Target | positive | *TP* | *FN* |
| | negative | *FP* | *TN* |

Figura 10. *Estructura de la Matriz de Confusión*

Al obtener valores altos en la diagonal que se forma de los verdaderos positivos y verdaderos negativos, entonces nuestro modelo es eficiente en cuanto a predicción de resultados.

Al aplicar la operacionalización en esta herramienta se puede obtener diferentes métricas de evaluación para los algoritmos de redes neuronales. Las más importantes son:

Tasa de clasificación errónea.

$$misclassification\ rate = \frac{(FP + FN)}{(TP + TN + FP + FN)}$$

Figura 11. *Fórmula para la Tasa de clasificación errónea*

La precisión que se obtiene en la clasificación.

$$classification\ accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

Figura 12. *Fórmula para la precisión del modelo*

Esto permite medir y saber que tan bueno es un modelo para predecir resultados.

(Jiawei Han, 2012) La metodología Knowledge Discovery in Databases (KDD) o también conocida como minería de datos se describe en el presente libro como un tema interdisciplinario porque hoy en día se aplica en diferentes campos. En el ámbito informático el término minería puede tener diferentes significados como extracción del conocimiento, análisis de datos, arqueología de datos. Pero comúnmente es conocido como descubrimiento de conocimiento.

Los autores consideran el proceso de descubrimiento del conocimiento como una secuencia de pasos que se representan y describen a continuación:

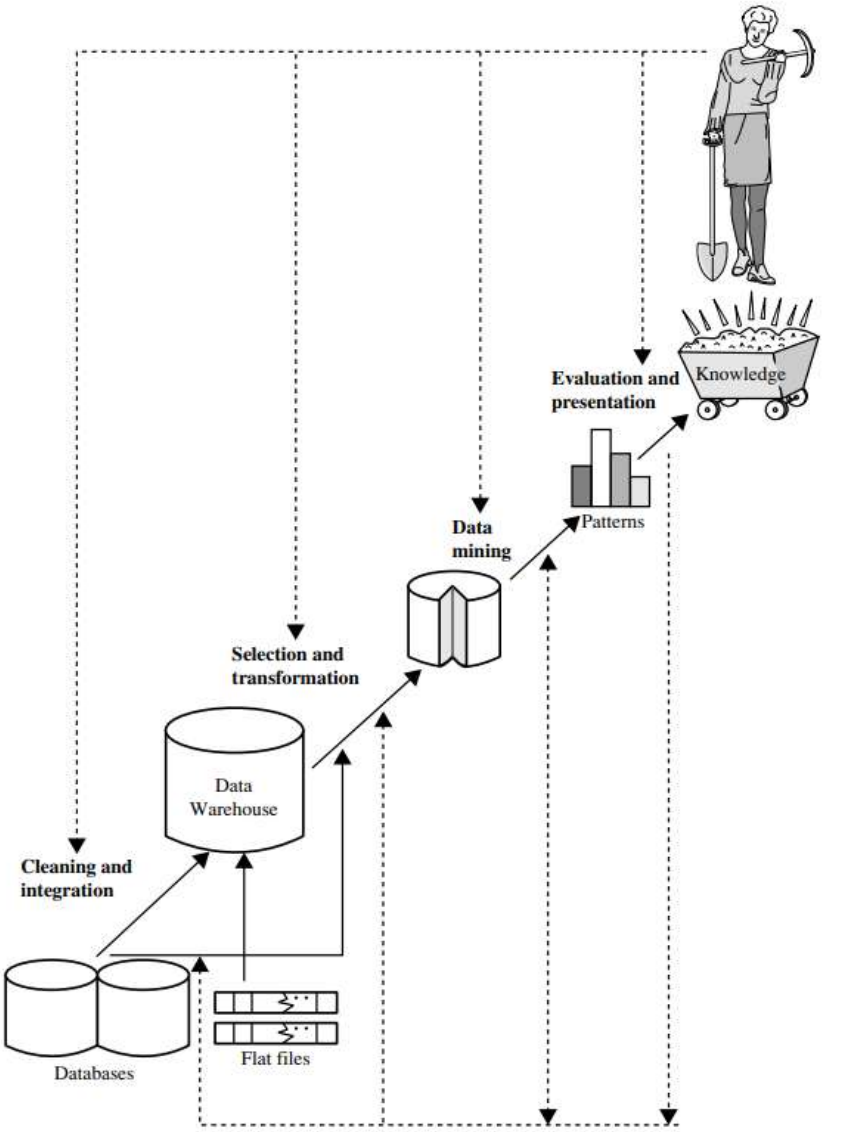



Figura 13. *Pasos para la extracción del conocimiento usando KDD*

1) Limpieza de datos

   En este paso se aplican técnicas, algoritmos o procesos que permiten eliminar ruido, datos incoherentes. Eliminando lo que no es importante y garantizando la permanencia de lo más útil.

2) Integración de datos

   Aquí se almacena todos los datos en un solo repositorio con la finalidad de homogenizarlos por categorías.

3) Selección de datos

   Fase donde los datos más relevantes son seleccionados para su procesamiento.

4) Transformación de datos

   Aquí los datos son procesados de una manera que permita convertirlos y consolidarlos en un aspecto adecuado para su posterior explotación.

5) Minería de datos

   Es el paso donde ya se aplican procedimientos o técnicas para la extracción de patrones.

6) Evaluación de patrones

   El punto donde se procede a identificar los patrones relevantes que representan el conocimiento que estamos buscando.

7) Presentación del conocimiento.

   Como último paso se procede a representar todo el conocimiento extraído mediante técnicas o herramientas de visualización que ayudan al usuario a comprender todo ese conocimiento explotado.

## 2.3. Marco Conceptual

### 2.3.1. Deep Learning

El Deep Learning es un subconjunto del aprendizaje automático, que en esencia se dedica al estudio y desarrollo de algoritmos con el objetivo que puedan aprender y llegar a converger en la inteligencia artificial. Aplicación mediante algoritmos como redes neuronales artificiales que son creado en inspiración al cerebro Humano con la finalidad de entrenarlos y hacerles de determinen o predigan un objeto. La siguiente figura muestra que el Deep Learning no se centra por completo en la Inteligencia Artificial. Si no que parte de esta se usar para resolver problemas en la industria. (Trask, 2019)

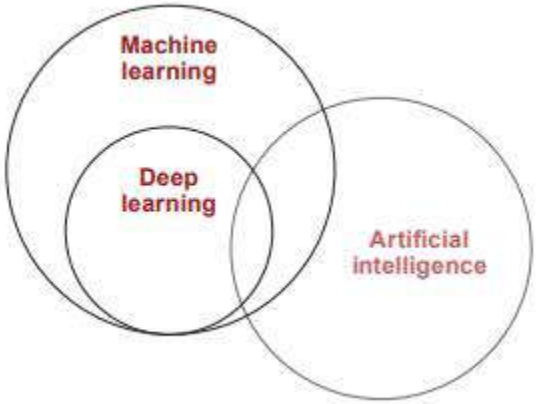



Figura 14. *Alcance del Deep Learning*

2.3.2. Plaga y Enfermedades en el cultivo de Maíz

2.3.2.1. Plagas del Maíz.

- Plaga: animal, planta o cualquier otro individuo que afecte, dañe u origine algún perjuicio al hombre, en sus animales, cosechas o sus posesiones. Se define así a todo ser vivo que se encuentra fuera del contexto. Es decir que no tiene nada que ver con el contexto o que este afectando aun si causa solo molestias puede ser catalogado como plaga. (Hill, 2007)

- Gusanos Cortadores: conocidos también como trozadores. Este tipo de insecto deja sus huevos por las noches en la maleza o el suelo. Los mismo que tiene un periodo de incuvacion de 4 a 6 días luego pasan a la etapa larvaria que dura 3 a 4 semanas para seguido pasar a su etapa pupal que dura un promedio de 15 días. Finalmente emergen como adultos con un periodo de vida de 6 a 8 días. Este tipo de plaga normalmente ataca por las noches en las partes mas bajas de la planta. (Vicente N. Páliz S.)

- Gusano Cogollero: Es una de las plagas de mayor importancia debido a grado de afectación principalmente en el cultivo de maíz. El origen de esta plaga se dio en los trópicos del Continente Americano. Desde que ya son unas larvas estas plagas ya pueden ocasionar leves daños superficiales en hojas ocasionando manchas blancas. Posteriormente al crecimiento de las larvas estas proceden a atacar el cogollo de la planta. En las plantas más pequeñas pueden llegar a ocasionarles la muerte. (Vicente N. Páliz S.)

- Gusano Choclero: Las larvas en los primeros estadios se alimentan y dañan los pistilos. Realizan pequeños agujeros para ingresar al interior de la mazorca, desde donde mastican el grano lechoso.
  Estos daños indirectos de la plaga son más destructivos que los causados por la larva, especialmente si el daño es abundante en lluvia. (Aroni, 2021)

- Pulgones: Son insectos chupadores con un pico largo que incrusta a la planta, también son conocidos con el nombre vulgar “Piojeras del Maíz” o “usa usa”. Constituye una plaga ocasional para las zonas andinas y valles interandinos. (Aroni, 2021)

- Cigarrita: Es una de las especies que se comporta como picadores chupadores de la savia de las plantas y se encuentra en las zonas productoras del maíz; mayormente en valles interandinos. La presencia de estas especies se registra durante todo el año. (Aroni, 2021)

- Barrenador del grano del maíz: Esta plaga es considerada en la mayoría de las zonas como plaga clave. El insecto se desarrolla en el interior del grano. Una vez en estado adulto se traslada de un grano a otro, el adulto vive 2 meses. (Aroni, 2021)

2.3.2.2. Enfermedades más comunes del Maíz

- Roya común: Esta enfermedad está ampliamente distribuida por todo el mundo, en climas subtropicales y templados y en tierras altas donde hay bastante humedad.
  En los estadíos iniciales de la infección tanto en el haz como envés de las hojas, la epidermis se rompe liberando un polvillo de color marrón oscuro, tornándose negra. (Aroni, 2021)

- Mancha parda: La incidencia de la enfermedad es mayor en terrenos cercanos a las riberas de los ríos, o en lotes con nivel freático alto y con tendencia al encharcamiento.
  Los esporangios liberan zoosporas, las cuales se mueven en el agua sobre la superficie de las hojas y atacan los tejidos más jóvenes, especialmente los de las hojas del cogollo. (Aroni, 2021)

- Carbón del maíz: Esta enfermedad se presenta sobre todo en lugares cálidos (25 a 30 grados centígrados) y algo secos. La enfermedad se puede presentar en todas las partes aéreas de la planta. (Aroni, 2021)

- Pudrición del cuello del tallo: Esta enfermedad ocurre en algunas zonas subtropicales o tropicales cálidas y húmedas, y en regiones templadas. Esta enfermedad puede afectar a las plantas antes de la floración. (Aroni, 2021)

- Pudrición del tallo: Las plantas marchitas permanecen erectas cuando se secan y aparecen lesiones pequeñas de color café oscuro en los entrenudos inferiores. (Aroni, 2021)

- Achaparramiento del maíz: En la zona andina esta enfermedad está causada mayoritariamente por, Spiroplasma kunkelii. Las hojas se

tornan a una tonalidad rojiza o púrpura, las mazorcas que se desarrollan son estériles y el sistema radicular también se reduce. (Aroni, 2021)

### 2.3.3. Imagen

Se define como una función bidimensional f(x,y) compuesta por dos variables independientes denominadas coordenadas. Relacionadas con un factor de intensidad que forman un objeto que su valor puede ser discreto o continuo. Dicha intensidad está compuesta por elementos(pixeles) que dan la forma a una imagen. (Woods, 2018)

### 2.3.4. Tipos de Imágenes

Basadas en su representación funcional:

**Imagen Binaria:** Se como aquella imagen que está compuesta de 0 y 1. Es decir, blanco (1) y negro (0). Por ello cada píxel necesita de 1 bit para almacenar su valor sea 0 o 1 que al ser mapeados en un eje de coordenadas formarán la imagen u objeto que se pretende visualizar. Por ejemplo, la siguiente figura compuesto por pixeles blancos y negros. (A Baskar, 2023)

Figura 15. *Pixeles Blancos y Negros*

**Imagen en Escala de Grises:** Se define como una imagen de tonalidades o intensidades en blanco y negro. Utiliza un formato de 8 bits, con una combinación de 256 tonalidades diferentes que van desde 0 que es el color negro, numeraciones intermedias con variaciones en intensidad y hasta 255 que viene a ser el color blanco. Por ello se denomina imagen en escala de grises porque van desde tonalidades de negro a blanco, lo que permite la formación de la imagen. (A Baskar, 2023)

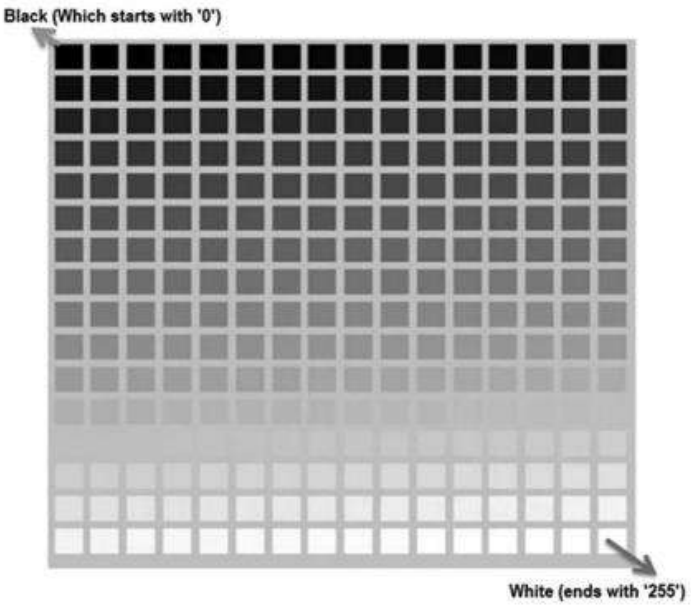



Figura 16. *Tonalidades en Escala de Grises*

**Imagen a Color:** Se define como una imagen compuesta de tonalidades o intensidades a color definidas por cada píxel y cada uno de los pixeles se encuentra formado por 3 canales que comúnmente se conocen como R, G y B donde sus siglas en ingles de cada canal representa su tonalidad en color Rojo, Verde y Azul. Miremos a una imagen como una matriz de pixeles como los pixeles se componen de los 3 canales y cada uno de los canales ocupa 8 bits. Entonces cada píxel ocupará un total de 24 bits, por ende, cada canal tendrá una tonalidad diferente de acuerdo con su imagen que represente. (A Baskar, 2023)

### 2.3.5. Píxel

Se define como la unidad o valor mínimo de una imagen digital. La conformación de pixeles en columnas y filas dan forma a una imagen. Es decir, una imagen se compone de pixeles que se encuentran adecuadamente adaptados para dar la forma a una imagen. (A Baskar, 2023)

### 2.3.6. Procesamiento de Imágenes

Proceso de realizar operaciones o cálculos con una imagen de entrada para mejorar o extraer la información que se quiere lograr obtener. Por ello lo recomendable es elegir las técnicas adecuadas para procesar la información obtenida de las imágenes. (A Baskar, 2023)

Finalidad del procesamiento de imágenes.

- ✓ Aumento de contraste. Para una mejor calidad de la imagen
- ✓ Compresión de imagen. Para reducir consumo de recursos.
- ✓ Mejorar imágenes borrosas, es decir restauración de imágenes.
- ✓ Identificación de objetos. Es decir, extraer determinadas características para su posterior uso. (Petrou, 2010)

### 2.3.7. Técnicas de Procesamiento de Imágenes.

Son métodos (Preprocesamiento de imagen, Mejora de Imagen, Segmentación de imagen, Extracción de características y Clasificación de Imágenes), que se aplican a datos bidimensionales que se encuentran formadas por una matriz de números. La que es representada por bits. (B. Chitradevi, International Journal of Innovative Research in Computer and Communication Engineering, 2014)

### 2.3.8. Términos en CNN

**Redes Neuronales:**

Procesador de conexión Inter neural de nodos distribuidos en paralelo y de forma masiva. Equipados con unidades de procesamiento simple con la finalidad de almacenar conocimiento para su posterior uso en el ámbito tecnológico. Tiene las siguientes 2 similitudes con el cerebro humano:

- ✓ conocimiento adquirido del aprendizaje.
- ✓ Cada conexión de una neurona tiene un valor (peso sináptico) lo que le permite almacenar el conocimiento aprendido. (Haykin, 2008)

**Convolución:**

se define como una operación matemática de matrices denominado integral del producto de dos funciones, luego de invertir y desplazar una sobre la otra función. (Maria Vakalopoulou, 2023)

$$h(t) = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,d\tau \quad .$$

Figura 17. *Integral del producto de 2 funciones*

**Aprendizaje Supervisado:**

Método que permite tener un conjunto elementos etiquetados como data de entrenamiento y que son usados por algoritmos de predicción. Donde permite identificar objetos no vistos con data etiquetada durante el entrenamiento. (Mehryar Mohri, 2018)

**Clasificación de Imagen:**

Es el proceso de categorizar cada píxel de una imagen en una clase que se asemeje a la imagen con la finalidad de identificar una imagen

por medio de sus características. Por ende, para una clasificación correcta la identificación entre datos y clases debe ser bien definidos. (Vinayak Bharadi, 2017)

**Localización de Objeto:**

El proceso trazar la ubicación de un objeto en una imagen en función a su eje central, su contorno del objeto, cuadro delimitador. Representación que se realiza base a sus coordenadas establecidas en un eje central. (Christoph H. Lampert, 2008)

**Segmentación Semántica:**

Es el proceso de etiquetar cada píxel con el propósito de clasificar a cuál clase pertenece dicho objeto que está presente. En función a los datos de la imagen local o datos de entrenamiento. Esta segmentación es una forma de agrupar o clasificar los objetos de manera más general. (Prince, 2010)

**Segmentación de Instancia:**

Es un procedimiento normalmente se basa en el método Mask R-CNN donde primero clasifica mediante el etiquetado, luego agrupa los pixeles delimitándolos para posteriormente obtener una máscara de instancia. (Beomyoung Kim, 2021)

**Función de Activación:**

Es una función Matemática que determina si una entrada en particular debe activarse o no. Es decir, propagar la salida de una neurona hacia la siguiente neurona de otra capa. (Josh Patterson, 2017)

**Capa Convolucional:**

Proceso que involucra aplicar filtros que recorren todos los pixeles de la imagen; para validar la existencia de características relevantes en la imagen y la obtención de una nueva matriz de pixeles producto de la

operacionalización entre el kernel y una parte de la imagen; con este proceso el kernel se mueve una cantidad de píxeles, lo que se denomina zancada. (Trask, 2019)

**Capa de Agrupación:**

Procedimiento que consiste en reducir la dimensionalidad del mapa de características por medio de cálculos matemáticos con la finalidad de mantener la información útil y eliminando la información irrelevante. La ventaja de la agrupación es reducción en la cantidad de parámetros a aprender, costos computacionales y mitiga la variación de posiciones de las características logrando mayor eficiencia en los modelos. (Hossein Gholamalinezhad, 2009)

**Capa Totalmente Conectada:**

Es la capa que recibe las entradas de las capas de convolución y agrupación. Salidas de estas capas que se encuentran en forma de vector o matriz unidimensional. Donde cada entrada de una capa (capas ocultas) está conectada a cada unidad de activación de la siguiente capa, lo que va a permitir mostrar el resultado final del modelo. (Rikiya Yamashita, 2018)

**Sobre Ajuste (Overfitting):**

Es el proceso de sobre aprendizaje. Es decir, cuando un modelo es entrenado por demasiado tiempo, puede ocasionar que el algoritmo no generalice bien su aprendizaje y esto provocaría el no ajustarse correctamente a los datos. Una acotación es no utilizar los datos de entrenamiento para pruebas porque daría predicciones erradas. (Ghayoumi, 2022)

**Bías:**

Se entiende como aquel valor definido como la constante que permite sumar al producto de características y pesos. Similar a la función

lineal y = mx + b con la finalidad de obtener mejores resultados al mover la función de activación hacia la izquierda o hacia la derecha. (Camila Laranjeira, 2021)

**HiperParámetros:**

Son variables definidas como constantes que se configuran antes del proceso de entrenamiento y sus valores contribuyen en el aprendizaje. El setteo de hiperparámetros se da de manera pragmática; es decir realizando pruebas de entrenamiento, validando los mejores resultados hasta encontrar el óptimo para el algoritmo del aprendizaje automático. (Nurshazlyn Mohd Aszemi, 2019)

**Parámetros:**

Son variables internas al modelo es decir sus valores no se configuran de manera manual. Por lo tanto, sus valores se obtienen a partir del entrenamiento con los datos proporcionados de las entradas y técnicas o funciones que se haya aplicado durante el entrenamiento. Son valores cambiantes durante el training. (Trask, 2019)

**Aumento de Datos:**

Se define como la técnica de aumentar la cantidad de datos utilizados para el entrenamiento del modelo. Con la finalidad de tener los datos suficientes para lograr que el modelo generalice de la mejor manera y obtener predicciones confiables. Algunas de las técnicas de aumento más usadas tenemos: (Ghayoumi, 2022)

- ✓ **Aumento de Posición**
  - Escalado
  - Recorte
  - Giro
  - Relleno
  - Rotación

- Transformación afín

✓ **Aumento de Color**

- Brillo
- Contraste
- Saturación
- Matiz

**Relu:**

Es una función de activación (Unidad Lineal Rectificada) que, al ingresar un valor positivo como entrada, mayor que cero se activará dicha neurona caso contrario desactiva la neurona. Esto permite no tener valores por debajo de cero. Considerando valores entre $[0, \infty\rangle$ eliminando valores negativos y reduciendo el problema de descenso de gradiente. (Ghayoumi, 2022)

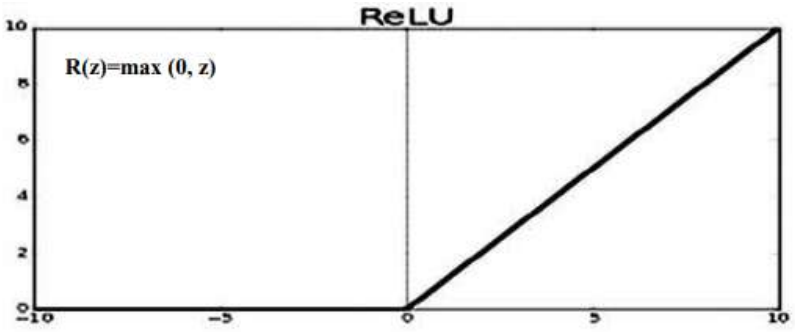



Figura 18. *Unidad Lineal Rectificada*

**Dense Layer:**

Es una capa que se encuentra completamente conectada. Es decir, las neuronas de la capa están conectadas a cada neurona de una capa anterior con la finalidad de que las neuronas densas realicen la multiplicación matriz – vector. (Harrison Kinsley, 2020)

**Función Fit:**

Es una forma para evaluar qué tan bien su modelo predice al entrenar y obtener resultados. Si las predicciones son erradas, su función generará un número más alto. Caso contrario obtendrá un número más bajo. (AI, 2018)

**Perceptrón:**

Se define como una red neuronal considerada como clasificador lineal por lo que es usado en el aprendizaje supervisado. Su desarrollo del perceptrón se da dentro de una neurona donde realiza ajuste en sus pesos y sesgos lo que le va a permitir realizar la clasificación de objetos de forma más eficiente. Normalmente la clasificación de patrones del objeto es separada en 2 a más clases de forma lineal. (Haykin, 2008)

**Dropout:**

Es el proceso que descartar o eliminar determinados nodos que no generalizan o aprenden bien. Con la finalidad de mejorar la generalización y el aprendizaje. Este proceso solo se aplica en las capas de entradas y capas ocultas. No en las de salida porque no permitiría obtener predicciones. (Ghayoumi, 2022)

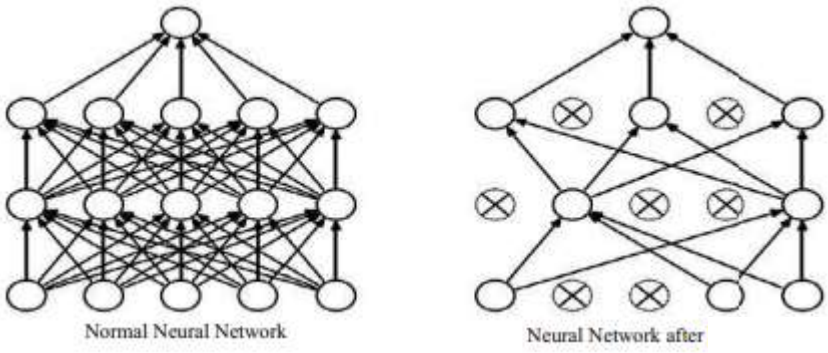



Figura 19. *Aplicación de Dropout*

**Seed:** Constante que define el grupo de datos que se toman en un proceso aleatorio.

**Weight:**

Los pesos son valores que operan en cada neurona y reflejan la importancia de las características en la predicción del resultado final. (Ganesh, 2020)

**Loss:**

Es la función que determina el valor promedio de las pérdidas de cada bach. Datos de entrenamiento. Los primeros lotes(baches) normalmente son mayores que los últimos. Cuando la predicción es buena, la pérdida es cercana o igual a cero; caso contrario es mayor.
Su objetivo es medir los resultados y los resultados esperados que se desea obtener. Esto permite evaluar su nivel de generalización de datos para realizar las predicciones. (Juan R. Terven, 2023)

**Val_Loss:**

Es la función que determina el rendimiento sobre un conjunto de datos de validación. El val_loss indica que tan bien se ajusta el modelo a los datos nuevos. A veces el valor de val_loss es mayor que loss y esto nos quiere decir que el modelo no se ajusta correctamente. (baeldung, 2022)

**Bach_Size:**

Es el número o cantidad muestras procesadas antes de actualizar el modelo y que tiene cada iteración de una época. (Aditya Devarakonda, 2018)

**Epoch:**

Se denomina así al procedimiento de iterar todo el ciclo de entrenamiento para cada conjunto de datos de entrenamiento. Eso quiere decir que toda una fase de entrenamiento lleva por nombre época. (Chollet, 2021)

**Iteración:**

es la cantidad de baches para completar una época. Es decir, todo el conjunto de datos. (SHARMA, Towards Data Science, 2017)

**Retropropagación:**

Es un algoritmo perteneciente al aprendizaje supervisado. Conocido como propagación hacia atrás o Backpropagation y su función es realizar cálculos matemáticos(gradientes) de algunos parámetros con el objetivo de mejor las predicciones en cada epoch. (Ghayoumi, 2022)

## 2.4. Sistema de Hipótesis

Un modelo de procesamiento de imágenes con Redes Neuronales. Detecta plagas y enfermedades en los cultivos de maíz en el Distrito de Cascas, La Libertad, durante el año 2022.

## 2.5. Variables e Indicadores

**VI:** Modelo de procesamiento de Imágenes con Redes Neuronales.

**VD:** Detección temprana de Plagas y Enfermedades en los cultivos de Maíz en el Distrito de Cascas, La Libertad, durante el año 2022.

## III. METODOLOGÍA EMPLEADA

| VARIABLE | DIMENSIÓN | INDICADOR | UNIDAD DE MEDIDA | INSTRUMENTO DE INVESTIGACIÓN |
|---|---|---|---|---|
| **VI** Modelo de procesamiento de Imágenes con Redes Neuronales. | Tiempo de procesamiento | Tiempo | Minutos | Reloj |
| | Precisión | Confiabilidad, Margen de Error | Porcentaje (%) | Redes Neuronales, Python y Librerías |
| **VD** Detección temprana de Plagas y Enfermedades en los cultivos de Maíz en el Distrito de Cascas, La Libertad, durante el año 2022. | Hoja con Plaga | Precisión | Porcentaje (%) | Modelo de Redes Neuronales |
| | Hoja con Enfermedad | Precisión | Porcentaje (%) | Modelo de Redes Neuronales |
| | Hoja sana | Precisión | Porcentaje (%) | Modelo de Redes Neuronales |

*Tabla 4. Matriz de Operacionalización*

### 3.1. Tipo y Nivel de Investigación

Tipo de Investigación es Longitudinal y por el diseño de contrastación es experimental y comparativa.

## 3.2. Población y Muestra de Estudio

### 3.2.1. Población

Cultivos de maíz con plagas y enfermedades, de los agricultores del distrito de Cascas, La Libertad, durante el año 2022.

### 3.2.2. Muestra

Cultivos de maíz con plagas, enfermedades y sanas de la parcela de la Sra. María Julia Alva León, ubicado en el caserío Cojitambo, Distrito de Cascas, La Libertad, año 2022.

## 3.3. Diseño de Investigación

| **LEYENDA** | |
|---|---|
| **Hojas Sanas** | GOP1 |
| **Hojas Enfermas** | GOP2 |
| **Hojas Con Plagas** | GOP3 |

*Tabla 5. Leyenda de grupo de estudio*

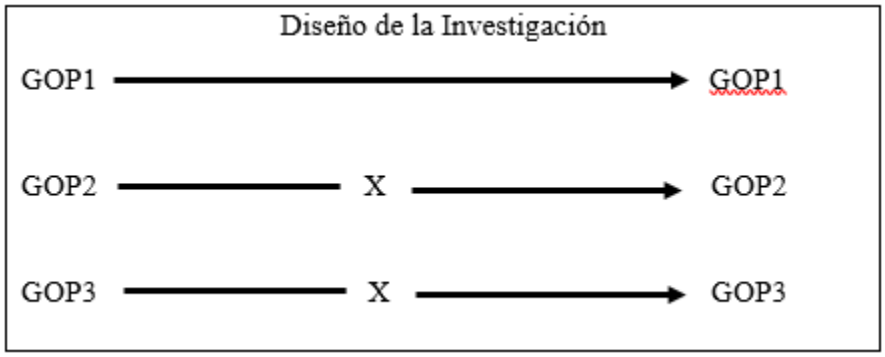



Figura 20. *Diseño de investigación*

## 3.4. Técnicas e Instrumentos de Investigación

- ✓ Adquisición de imágenes mediante teléfono móvil.

- ✓ Observación.
- ✓ Internet

## 3.5. Procesamiento y Análisis de Datos

Luego de recopilar el Dataset de imágenes y haber entrenado a cada modelo. Se procede a medir el rendimiento de cada uno.

Para medir el rendimiento de cada modelo se utilizará una Matriz de Confusión. Herramienta que permite analizar los resultados, haciendo uso del etiquetado y clasificación multiclase. La misma que nos brinda una representación visual indicando en que se confunde nuestro modelo. Donde las filas representan valores reales. Es decir, imágenes desconocidas por el modelo y las columnas son valores conocidos (información obtenida durante el entrenamiento). Datos con los que va a comparar para medir que tan preciso es un modelo.

| | | **Predicted** | |
|---|---|---|---|
| | | Negative **(N)** - | Positive **(P)** + |
| **Actual** | Negative - | True Negative **(TN)** | **False Positive (FP) Type I Error** |
| | Positive + | **False Negative (FN) Type II Error** | True Positive **(TP)** |

Figura 21. *Métricas de la Matriz de Confusión.*

La interpretación de un buen modelo se da cuando tiene valores altos en TN, TP; y valores mínimos en FP y FN. (Suresh, Medium, 2020)

## IV. PRESENTACIÓN DE RESULTADOS

Aplicación de la Metodología:

Para nuestro proyecto se trabajó con modelos pre entrenados desde la base de datos ImageNet. La misma que contiene millones de imágenes en diferentes ángulos con la finalidad de aplicarlos en modelos de inteligencia artificial logrando mejorar los resultados de identificación. Cabe decir que las imágenes son de diferentes especies, animales, cosas, plantas, etc.

Esto permite que los modelos a seleccionar obtengan resultados en menor tiempo, menos recursos computacionales, y menos horas hombre. Lo cual trae consigo excelentes predicciones en comparación a resultados de años anteriores.

| MODELOS SELECCIONADOS | | | | |
|---|---|---|---|---|
| MODELO | TAMAÑO (MB) | PRECISIÓN (TOP 1) | PRECISÓN (TOP 5) | PARÁMETROS (Millones) |
| **ResNet50** | 98 | 76.0 % | 93.0% | 25.6 |
| **DenseNet201** | 80 | 77.3% | 93.6% | 20,2 |
| **EfficientNetV2B0** | 29 | 78.7% | 94.3% | 7.2 |
| **MobileNetV2** | 14 | 71.3% | 90.1% | 3.5 |

*Tabla 6. Modelos Seleccionados*

Librerías aplicadas para el desarrollo de los modelos.

| LIBRERIAS | |
|---|---|
| **Keras** | Es una biblioteca de código abierto que permite trabajar con otras librerías de manera intuitiva para la creación de redes neuronales. |
| **Tf** | Es una biblioteca de tensorflow desarrollada para realizar seguimiento a los marcos de coordenadas y transformar los datos (puntos,vectores...) de un sistema. Esto permite correrlo con otros componentes en conjunto. Asegurando el marco correcto. |
| **Layers** | Librearía que permite construir conjunto de capas funcionales en modelos de keras. |
| **Regularizers** | Es una forma de calibrar los modelos mediante los parámetros de entrada con los coeficientes más grandes. Con el fin de minimizar la función de pérdida y evitar el overfitting / underfitting. |
| **DateTime** | Librería propia de python que se usa para manipular datos en formato fecha y hora. |
| **Os** | Librería que permite interactuar con el sistema operativo. Accediendo a realizar operaciones como crear, buscar, cambiar, eliminar directorios. |
| **Yaml** | Es un recurso para serializar los datos legibles para las personas y que permite realizar configuraciones en archivos. |

*Tabla 7. Librerías utilizadas en el Proyecto*

## 4.1. Análisis e Interpretación de Resultados

### 4.1.1. Selección de Imágenes.

Para el entrenamiento de los modelos se utilizó imágenes agrupados en 4 categorías conformadas por un total de 5158 de las que 4127 se usan para entrenamiento (80%) y 1031 para validación (20%)

**Categorías:**

A. Tizón de la hoja de maíz (Blight)
B. Roya común (Common Rust)
C. Gusano (Armyworm)
D. Sanas (Healthy)

### 4.1.2. Pre Procesamiento

a) Se crea la estructura de las carpetas para almacenar las imágenes, notebooks con extensión Python .ipynb en las que se crea los modelos, para lograr el objetivo de esta Tesis.

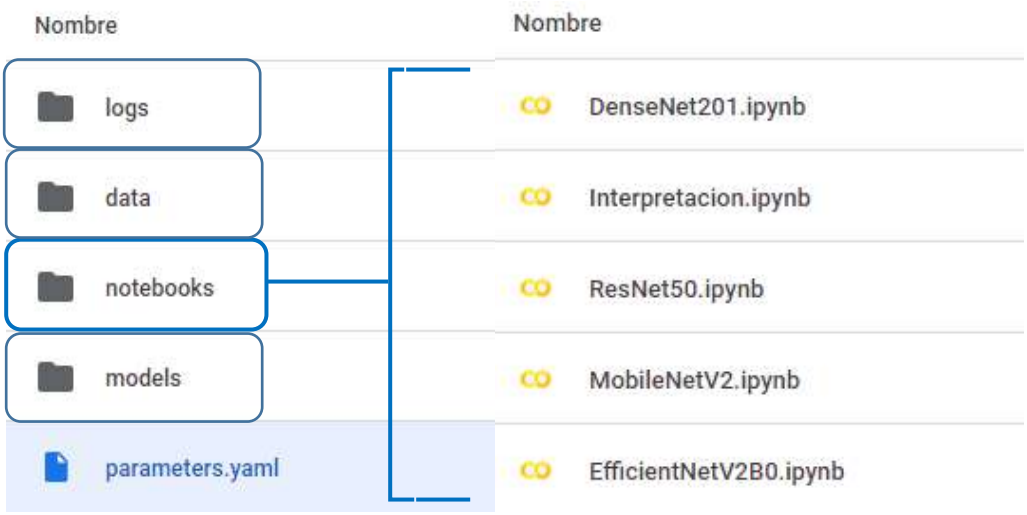



Figura 22. *Notebooks para procesamiento de Imágenes de Maíz*

La carpeta logs (Almacena el resultado del entrenamiento de cada modelo. Para luego utilizarlo en la predicción de una imagen nueva.) y models (Guarda el resultado de varias versiones, cuando un epoch que es superado por otro en el nivel de accuracy, finaliza guardando la última versión) respectivamente.

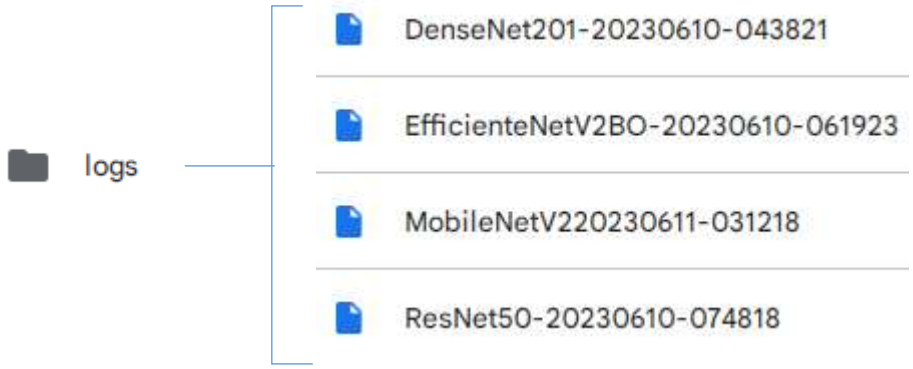



Figura 23. *Almacena la información aprendida durante el entrenamiento*

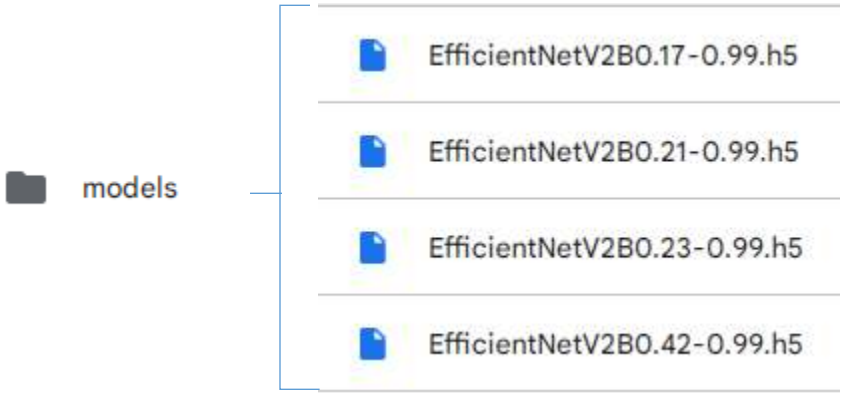



Figura 24. *Almacén de los mejores resultados para cada modelo*

Y la carpeta data (contiene subcarpetas con las imágenes para el entrenamiento).

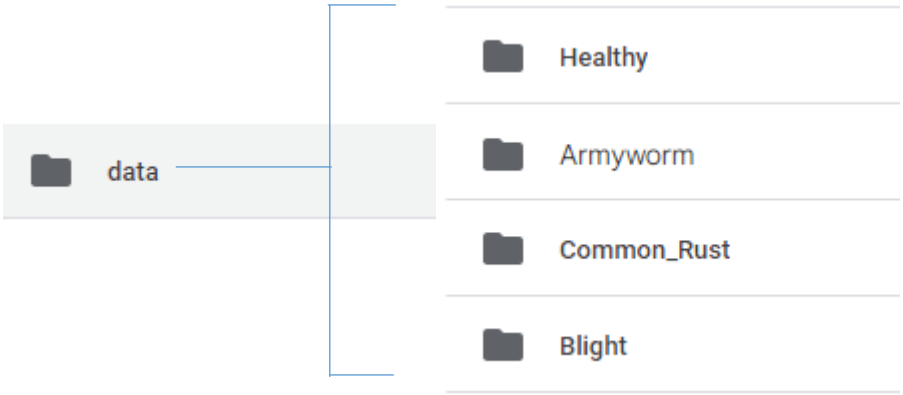



Figura 25. *Clasificación de Imágenes para Entrenamiento de Modelos*

b) Se crea un archivo con extensión .yaml que va permitir construir el archivo de configuración de parámetros similar a JSON.

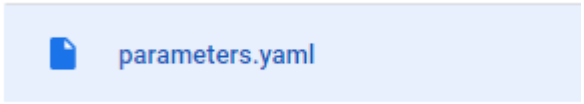



Figura 26. *Archivo de configuración de parámetros*

c) Se definen las variables dentro del archivo parameters.yaml que van a configurar los tamaños de ingreso de las imágenes, técnicas para mejorar la generalización de los modelos, funciones y constantes que van a ser usadas dentro del desarrollo del modelo; para crear y entrenar el modelo de red neuronal.

Parámetros:

seed: Es el número que se define para inicializar el proceso aleatorio de training y validación donde va a tomar el mismo conjunto de datos de la secuencia anterior, en cada iteración de ambos procesos.

raw_images: Ruta que acopia las imágenes para el entrenamiento.

Image_Size: 240 * 240 Tamaño de la imagen para cada neurona

batch_size: 32 Cantidad de imágenes que se van a propagar en la red neural (número de muestras)

buffer_size: 32 Número de imágenes que se almacenan de la propagación.

validation_split: 0.2 Porcentaje de imágenes que se toman para la validación.

```yaml
global:
  seed: 42
data:
  raw_images: "/content/drive/MyDrive/project/data/unzipped/data/"

preprocesamiento:
  image_size: !!python/tuple [240, 240]
  batch_size: 32
  buffer_size: 32
  validation_split: 0.2

utils:
  muestreo_imagenes:
    image_size: !!python/tuple [10, 10]
    num_imagenes: 9
augmentation:
  random_flip:
    # https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomFlip
    # String indicating which flip mode to use.
    # Can be "horizontal", "vertical", or "horizontal_and_vertical".
    # Defaults to "horizontal_and_vertical".
    # "horizontal" is a left-right flip and "vertical" is a top-bottom flip.
    mode: "horizontal"
  RandomRotation:
    # https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomRotation
    fill_mode: "reflect"
    interpolation: "bilinear"
    factor: !!python/tuple [-0.1, 0.1]
  RandomZoom:
    # https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomZoom
    height_factor: !!python/tuple [-0.1, 0.1]
    width_factor: !!python/tuple [-0.1, 0.1]
```

Figura 27. *Parámetros configurados en archivo parameters.yaml*

Cada uno de los parámetros configurados a partir de **Utils** se irán explicando conforme el desarrollo del informe.

```yaml
model:
  batch_normalization:
    momentum: 0.99
    epsilon: 0.001
  dense_1:
    units: 256
    l2: 0.016
    l1: 0.006
    bias_regularizer: 0.006
    activation: "relu"
  dropout:
    rate: 0.45
  dense_2:
    activation: "softmax"
callbacks:
  early_stopping:
    patience: 3
  reducelronplateau:
    monitor: "val_loss"
    factor: 0.2
    patience: 2
    min_lr: 0.00001

compile:
  lr: 0.001

save_model:
  path: "/content/drive/MyDrive/project/models"

logs:
  path: "/content/drive/MyDrive/project/logs"
```

Figura 28. *Configuración de callbacks*

d) Ya en el archivo de cada modelo se importan las librerías.

```
[ ]  import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras import layers
     from tensorflow.keras import regularizers

     import datetime, os
     import yaml
```

Figura 29. *Importación de librerías keras*

e) Accedemos a las carpetas creadas en Google Drive para llevar a cabo proceso de cada modelo.

```
from google.colab import drive
drive.mount('/content/drive')
```

Figura 30. *Acceso a data almacenada en google drive*

f) Se carga los parámetros configurados en el archivo parameters.yaml y se lista las categorías.

```
with open("/content/drive/MyDrive/project/parameters.yaml") as conf_file:
  config = yaml.full_load(conf_file)
categories = os.listdir(config["data"]["raw_images"])
n_categorias = len(categories)
print(f"Tenemos {n_categorias} categorías:\n
{[category for category in categories]}")
```

```
Tenemos 4 categorías:
['Healthy', 'Blight', 'Common_Rust', 'Armyworm']
```

Figura 31. *Listar categorías de imágenes.*

g) Se llama los parámetros como tamaño de la imagen, número de imágenes para cada iteración, el porcentaje de imágenes para validación, la semilla para el inicializar en el mismo valor del proceso aleatorio en training como en validation.

El proceso de dividir el conjunto de imágenes para entrenamiento y validación con el componente de la librería keras **image_dataset_from_directory** seteando los valores para cada nodo.

### Nodos de Entrenamiento y Validación

```
image_size = config["preprocesamiento"]["image_size"]
batch_size = config["preprocesamiento"]["batch_size"]
validation_split = config["preprocesamiento"]["validation_split"]
seed = config["global"]["seed"]

# Node training_set
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    config["data"]["raw_images"],
    validation_split=validation_split,
    subset="training",
    seed=seed,
    image_size=image_size,
    batch_size=batch_size,
)
# Node validation_set
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    config["data"]["raw_images"],
    validation_split=validation_split,
    subset="validation",
    seed=seed,
    image_size=image_size,
    batch_size=batch_size,
)
```

Figura 32. *Creación de nodos de entrenamiento y validación*

| IMÁGENES ESTABLECIDAS POR CADA NODO | |
|---|---|
| Nodo de Entrenamiento (80%) | 4127 |
| Nodo de Validación (20%) | 1031 |
| Total, de Imágenes (100%) | 5158 |

*Tabla 8. Número de Imágenes asignadas a cada nodo*

Donde de 4 clases (categorías de las imágenes) con un total de 5158 en formato .jpg 4127 fueron seleccionadas para entrenamiento y 1031 para validación.

| CATEGORÍA | ETIQUETA |
|---|---|
| Gusano | **0** |
| Tizón de la Hoja de Maíz | **1** |
| Roya Común | **2** |
| Saludable | **3** |

*Tabla 9. Etiquetado de grupo de estudios*

h) Se muestra las imágenes etiquetadas por un número de acuerdo con la categoría. Utilizando la librería de Matplotlib. En un gráfico de 9 imágenes con tamaño de 10 x 10.

```
# node muestreo_imagenes
import matplotlib.pyplot as plt

plt.figure(figsize=config["utils"]["muestreo_imagenes"]["image_size"])
for images, labels in train_ds.take(1):
    for i in range(config["utils"]["muestreo_imagenes"]["num_imagenes"]):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(int(labels[i]))
        plt.axis("off")
```

Figura 33. *Líneas de código para mostrar imágenes redimensionadas.*

Imágenes etiquetadas con gusano, tizón de la hoja, roya común y sana.

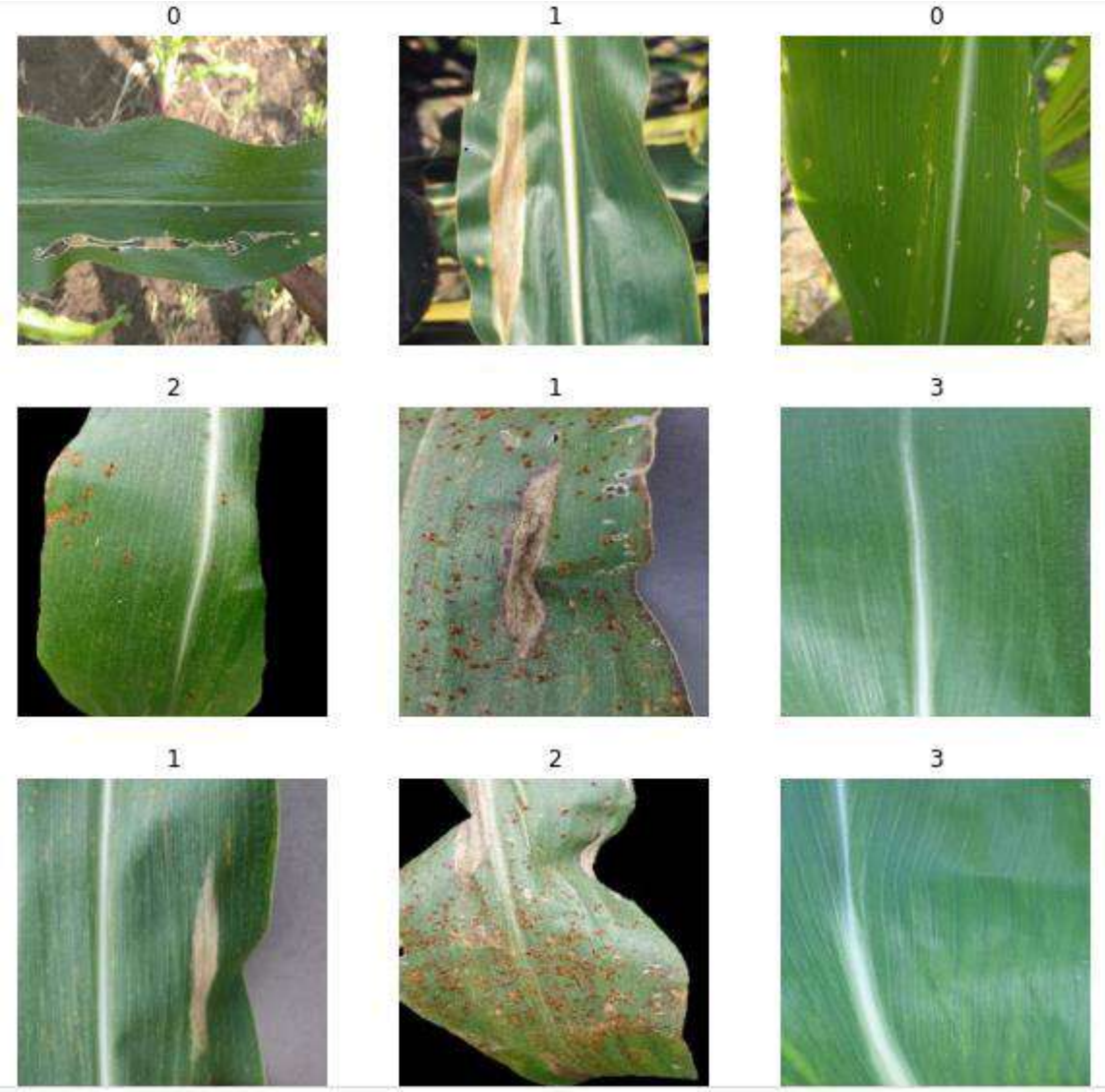



Figura 34. *Imágenes etiquetadas*

i) Se aplica técnicas Augmentation para mejor la generalización y la tasa de aprendizaje. Estos parámetros son seteados en el archivo paremeters.yaml para luego ser usados dentro la construcción del modelo.

   Las técnicas de aumento de datos se aplican normalmente cuando no se tiene un dataset de grandes proporciones; como una buena práctica.

   Las técnicas utilizadas son:

**Random_flip:** En este caso voltea la imagen de izquierda a derecha de manera aleatoria.

**RandomRotation:** Permite aplicar rotaciones aleatorias a cada imagen durante el entrenamiento. Algunos de los argumentos configurados son:

i. Fill_mode "reflect" : la entrada se amplía reflejando el borde del último píxel.

ii. Interpolation : Permite dar un mayor tamaño a la imagen inicial, rellenando los espacios vacíos con nuevos datos calculados a partir de un algoritmo específico. En este caso se usa: **Bilinear** se basa en promediar los 4 pixeles adyacentes para conseguir la información a rellenar.

iii. Factor : Los valores representados en una tupla en Python se usan para la rotación en sentido horario y antihorario. Valor positivo quiere decir que gira en sentido contrario a las agujas del reloj, mientras que un valor negativo gira en sentido a las agujas del reloj.

**RandomZoom:** Permite incrementar aleatoriamente el tamaño de la imagen se utilizó dos de sus argumentos principales:

i. Height_factor : Los valores de la tupla hacen referencia al límite inferior y superior para hacer zoom verticalmente.

ii. Width_factor : Los valores de la tupla hacen referencia al límite inferior y superior para hacer zoom horizontalmente

Proceso de aumentación de las imágenes.

```
# node augmentation
data_augmentation = keras.Sequential(
    [
        # # Rotación de imágenes horizontal y vertical
        layers.RandomFlip(mode=config["augmentation"]["random_flip"]["mode"]),
        # Rotación
        layers.RandomRotation(fill_mode=config["augmentation"]["RandomRotation"]
                              ["fill_mode"],
        interpolation=config["augmentation"]["RandomRotation"]["interpolation"],
        factor=config["augmentation"]["RandomRotation"]["factor"]),
        layers.RandomZoom(height_factor=config["augmentation"]["RandomZoom"]
                          ["height_factor"],
        width_factor=config["augmentation"]["RandomZoom"]["width_factor"]),
    ]
)
```

Figura 35. *Funciones de aumentación*

En el nodo de aumentación de datos se aplican las técnicas repetidamente en cada capa. Luego de eso se visualiza las imágenes con las técnicas aplicadas antes mencionadas.

```
# node
plt.figure(figsize=config["utils"]["muestreo_imagenes"]["image_size"])
for images, _ in train_ds.take(1):
    for i in range(config["utils"]["muestreo_imagenes"]["num_imagenes"]):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```

Figura 36. *Recorrido de imágenes para visualizar*

Aumentación de Imágenes.

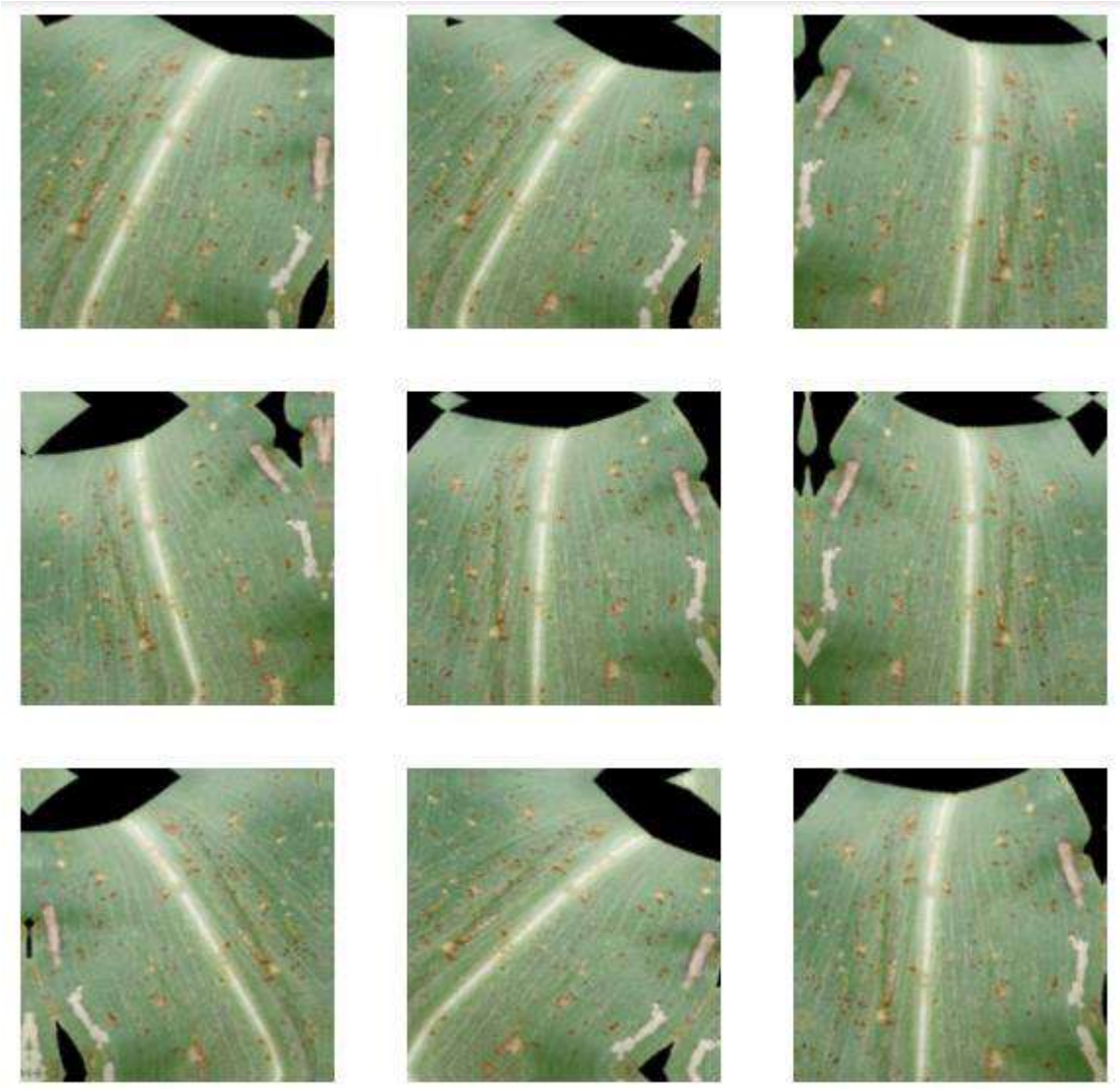


Figura 37. *Imágenes aumentadas*

j) Se pasan las imágenes utilizadas en la propagación de la red neuronal (imágenes precargadas en la red). Utilizando la función prefetch se va a obtener las imágenes propagadas. Luego se almacena en variables que representan el nodo de entrenamiento y validación que van a ser usadas más adelante.

Propagación de imágenes.

```
buffer_size = config["preprocesamiento"]["buffer_size"]

# node
train_ds = train_ds.prefetch(buffer_size=buffer_size)
# node
val_ds = val_ds.prefetch(buffer_size=buffer_size)
```

Figura 38. *Obtener imágenes propagadas*

k) Una vez aplicado técnicas y funciones de pre procesamiento. Se procede a crear el Modelo mediante la función **make_model.**

```
# node
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    # Image augmentation block
    x = data_augmentation(inputs)

    # Entry block
    x = layers.Rescaling(1.0 / 255)(x)

    # Pretrained model
    base_model = keras.applications.DenseNet201(
        include_top=False,
        weights="imagenet",
        input_shape=input_shape)
```

Figura 39. *Creación de modelo DenseNet201*

- ✓ Se define una función donde se va a pasar 2 parámetros el **input_shape** representa al valor de cada imagen y el **num_classes** que vienen a ser las 4 categorías de imágenes. Luego se aplica las técnicas de aumentación para obtener una mejor generalización en el entrenamiento, lo que va a permitir que un modelo sea mucho más preciso.
- ✓ La siguiente línea realiza un reescalado de las imágenes de 1 a 255 por las imágenes aumentadas. Logrando con ello obtener valores entre 0 y 1 para que al algoritmo se le haga más fácil leerlos. Todo ello considerando los 3 canales que representan a RGB.

- ✓ Ahora pasamos a definir el modelo y sus parámetros. Donde **include_top** si está en true toma la salida de la clasificación de **Imagenet** es decir de la cabecera; pero en la presente investigación se considera false para obtener la salida de la clasificación de las 4 categorías consideradas para nuestro estudio y los pesos se toman de **Imagenet** para aplicarlos en los modelos seleccionados.
- ✓ Los datos aumentados, reescalados y aplicados el modelo seleccionado se le pasa a la variable que referencia al modelo **base_model.**
- ✓ En la capa **Flatten** tomamos la salida del **base_model** que trae la data (conjunto de características de cada imagen) en forma de matriz multidimensional y la convertimos en unidimensional. Con el fin de mejorar el entrenamiento de las capas debido a que la estructuras de las mismas es dimensional y esto le facilita en el aprendizaje.

```
outputs = base_model(x)
outputs = layers.Flatten()(outputs)
outputs = layers.BatchNormalization(axis=-1,
                                    momentum=config["model"]
                                    ["batch_normalization"]
                                    ["momentum"],
                                    epsilon=config["model"]
                                    ["batch_normalization"]
                                    ["epsilon"])(outputs)
outputs = layers.Dense(config["model"]["dense_1"]["units"],
                       kernel_regularizer =
                       regularizers.l2(l = config["model"]
                                       ["dense_1"]["l2"]),
                       activity_regularizer =
                       regularizers.l1(config["model"]
                                       ["dense_1"]["l1"]),
                         bias_regularizer =
                         regularizers.l1(config["model"]
                                         ["dense_1"]
                                         ["bias_regularizer"]),
                         activation = config["model"]["dense_1"]
                         ["activation"])(outputs)
```

Figura 40. *Normalización de imágenes*

- ✓ La siguiente capa se encarga de normalizar los datos. Es decir, calculando la media y la varianza vamos a lograr que los valores de las características de los datos ingresados oscilen en el rango de 0 y 1 pero que no sobrepasen mayor o menor. Logrando tener valores en la misma escala.
- ✓ La capa densa es la que se encuentra completamente conectada. La misma que cada neurona recibe como entrada la salida de cada neurona de la capa anterior. En la capa densa se definen determinados parámetros que van a ayudar en la generalización de los datos. Units es el parámetro que se define como un entero, para representar la dimensión de la salida en este caso es de 256. Para estos parámetros de regularización se opera con un valor de peso cercano a 0 para controlar valores altos en el peso y que cada neurona tenga como salida números grandes y evitar el sobreajuste en la etapa de entrenamiento.

- ✓ Kernel_regularizer es la variable que va a recibir el resultado del cálculo de L2 que representa aquellas características menos importantes en una hoja o planta de maíz. Que van a permanecer teniendo influencia en el entrenamiento del modelo. y que obliga a que los a valores de las características no sean igual a cero. Es decir, tenga valores bajos pero diferentes de cero.

- ✓ Activity_regularizer recibe los valores de L1 que viene a transformar aquellas características sin importancia para la predicción, como las partes ruidosas de las imágenes, en cero. con la finalidad de evitar el sobreajuste. Cuando se hace mención a ruido nos referimos aquellas partes de la imagen que no tienen nada que ver con la planta u hoja de maíz es por ello la aplicación de este tipo de técnicas de regularización. El valor considerado en L1 se define como 0.006 es decir menor a L2.

- ✓ Bias_regularizer representa el valor establecido que se aplicó al vector bias con el objetivo de regularizar la salida de la función.

- ✓ Activation que va a recibir la función de activación **Relu** para determinar si se activa o no una neurona.
  Todos los parámetros de la capa densa van a operar con la salida de la capa de Normalización.

```
outputs = layers.Dropout(rate = config["model"]
                         ["dropout"]["rate"],
                         seed = config["global"]
                         ["seed"])(outputs)
outputs = layers.Dense(n_categorias,
                       activation=config["model"]
                       ["dense_2"]["activation"])(outputs)

return keras.Model(inputs, outputs)
```

Figura 41. *Recepción de datos normalizados en capa Densa*

- ✓ Otra de las capas definidas el Dropout. Con ella logramos eliminar el ruido en cada iteracción; es decir aquellas zonas de la imagen que carecen de relevancia para nuestro propósito. Esto nos ayuda a evitar el sobre ajuste y mejorar la generalización de los modelos. Tal es el caso que durante la fase de entrenamiento y las imágenes sean distintas a las ya conocidas; el modelo pueda inferir de una manera eficiente.
  Para el modelo se define los siguientes argumentos: **rate** = 0.45. Que en términos de porcentaje vendría a eliminar el 45% de las neuronas que tienen ruido. Lo recomendable para redes grandes es considerar un valor de 0.5. Pero dado nuestro caso solo se considera el 0.45. El otro argumento es **Seed** el cual se encuentra seteado con un valor entero de 42 con el objetivo de tomar el mismo conjunto de características debido a que es un proceso aleatorio donde no solo se va cambiar el modelo de cnn sino también algunos otros parámetros y estos nos va permitir operar sobre el mismo conjunto de datos.

- ✓ La última capa establecida en la creación del modelo es otra capa densa que va a ejecutar la función **softmax,** la misma que va a convertir los valores de las características en valores reales que van oscilar entre 0 y 1. Los que obtengan valores mayores dentro de ese rango van a permitir determinar la salida(predecir). La última capa va a operar con la salida de las capas antes descritas.

```
# node
model = make_model(input_shape=image_size + (3,), num_classes = n_categorias)
# logging
keras.utils.plot_model(model, show_shapes=True)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/
74836368/74836368 [==============================] - 0s 0us/step
```

Figura 42. *Llamado de función con sus parámetros*

Finalmente se retorna la función considerando los parámetros **image_size** más los **3** canales RGB y **n_categorias** el número de categorías que se tomaron como muestra para llevar a cabo el proyecto. Luego de haber creado el modelo lo que se hace es visualizar la estructura del modelo can una de las capas definidas y sus respetivas entradas y salidas considerando los elementos especificados.

Estructura del Modelo

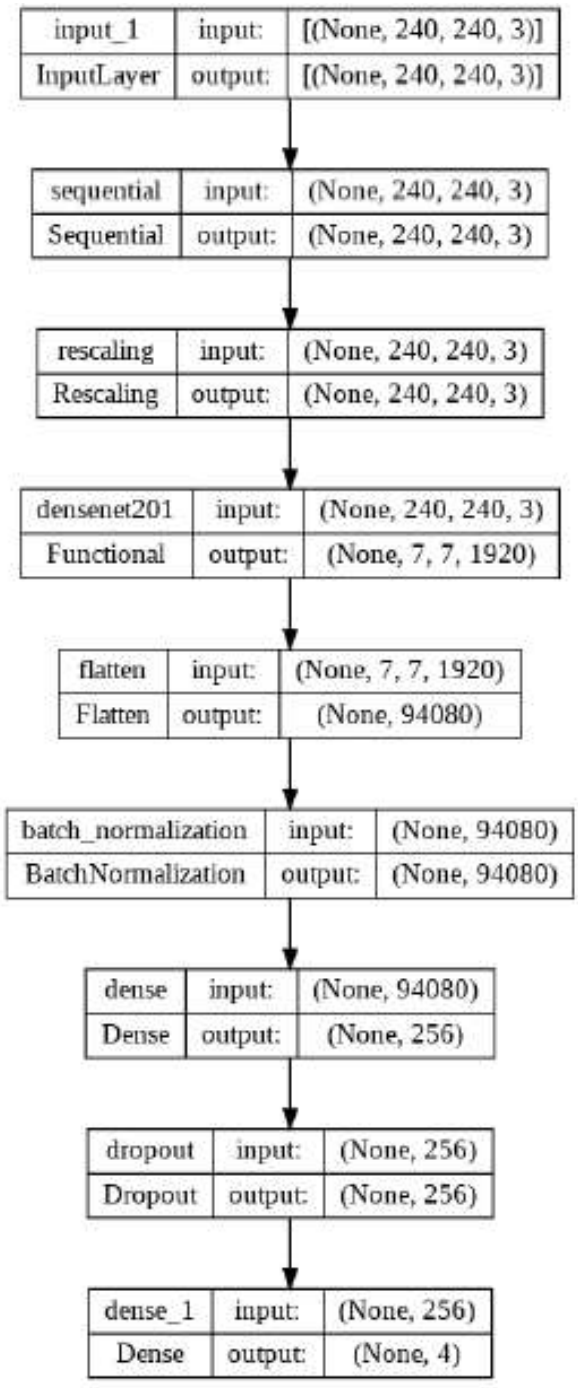



Figura 43. *Estructura de capas del modelo*

### 4.1.3. Entrenamiento y Validación.

En esta etapa se procede a entrenar el modelo primero se define el número de épocas(epoch) por las que van a pasar todo el conjunto de imágenes para ser entrenadas por el modelo. Es decir 50 veces verá a todas las imágenes cada modelo seleccionado.

En la variable **my_callbacks** se va a aplicar funciones con el objetivo de controlar algunas acciones del entrenamiento.

Se define el argumento de la función **early_stopping** con un valor de **patience** = 3, es decir con esto se le dice que luego de 3 épocas que deja de mejorar se detenga el entrenamiento. La segunda función definida es **ReduceLROnPlateau,** el fin de esta función es reducir la tasa de aprendizaje cuando el modelo ha dejado de mejorar alguna de las métricas relacionadas con la precisión. Sus argumentos definidos son:

**Monitor:** Representa la cantidad a ser monitoreada y es un valor tomado de la función de validación de pérdida.

**Factor:** Este argumento va a permitir reducir la tasa de aprendizaje, este valor normalmente se define en un rango de 0 a 1. Para fines de nuestro desarrollo tiene un valor de 0.2.

**Patience:** Es la cantidad de épocas que van a transcurrir sin mejorar. Para luego proceder a reducir la tasa de aprendizaje.

**Min_lr:** Este valor definido es el valor más bajo que puede tener una tasa de aprendizaje.

```
epochs = 50

log_dir = config["logs"]["path"] + datetime.datetime.now()
.strftime("%Y%m%d-%H%M%S")

my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=config["callbacks"]
                                     ["early_stopping"]["patience"]),
    tf.keras.callbacks.ReduceLROnPlateau(monitor=config["callbacks"]
                                         ["reducelronplateau"]["monitor"],
                                         factor=config["callbacks"]
                                         ["reducelronplateau"]["factor"],
                                         patience=config["callbacks"]
                                         ["reducelronplateau"]["patience"],
                                         min_lr=config["callbacks"]
                                         ["reducelronplateau"]["min_lr"]),
    tf.keras.callbacks.ModelCheckpoint(filepath=os.path.join(
        config["save_model"]["path"],
        "DenseNet201.{epoch:02d}-{val_accuracy:.2f}.h5"),
                                       monitor="val_accuracy",
                                       mode="max",
                                       save_best_only=True),]
```

Figura 44. *Setteo de epochs y funciones callbacks*

Una tercera función es **ModelCheckpoint** donde se definieron los argumentos el primero que es la ruta **path** donde se va a guardar el modelo entrenado denominando a cada modelo con su *nombre, la época, la precisión de validación y su extensión*. **Monitor** argumento que representa los valores a considerar para el procedimiento de la función. En este caso se toman los de val_accuracy.

**Mode** se establece en max porque los valores considerados a monitorear son del val_accuracy.

**Save_best_only:** Se estableció en valor ‘true’ para no sobre escribir el ultimo mejor modelo obtenido del entrenamiento.

En la última parte del entrenamiento lo que se hace es configurar el modelo con la palabra reservada **compile,** pasándole los siguientes argumentos:

**Optimizer:** donde le va a indicar la distancia que va recorrer, para minimizar el error y obtener aquel valor mínimo. El lr (learning rate) se

consideró valor predeterminado 0.001. Lo que permite aplicar el descenso de gradiente. Para obtener el valor esperado y que los resultados de predicción de plagas sean mucho mejor.

El segundo argumento es **loss,** en donde se aplica la función **SparseCategoricalCrossentropy** la misma que permite etiquetar las clases con números enteros logrando con ello optimizar el uso de recursos. Por lo mismo que nuestros datos son multiclase la mencionada función se ajusta a nuestro objetivo. Dentro de la mismas definimos un argumento como valor booleano 'False'. Lo que significa si estuviera en valor True, que no estuviésemos usando función softmax. Pero para nuestro propósito si aplicamos softmax por ese motivo se define como False. Y por último la métrica que define como evaluar el resultado del entrenamiento.

```
model.compile(
    optimizer=keras.optimizers.Adamax(learning_rate=config["compile"]["lr"]),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=["accuracy"],
)
history = model.fit(train_ds, epochs=epochs,
          callbacks=[my_callbacks], validation_data=val_ds,)
```

Figura 45. *Compilado del Modelo*

En la definición del método **model.fit** se procede a **entrenar** el modelo aplicado, ingresándole argumentos que contienen la data y las funciones antes configuradas que fueron almacenadas en variables y que ahora se pasan como argumento de model.fit.

En la siguiente imagen se muestra el proceso de entrenamiento donde: Epoch = 50 quiere decir iterar 50 veces o hasta el epoch que detecte una mejor precisión.

El número 129 se calcula del 80% de imágenes tomadas para el entrenamiento que hacen un total de 4127 y que se van a dividir en bach_size = 32

**Épocas e Iteraciones de los Modelos Entrenados.**

**A. Ciclo de entrenamiento del Modelo DenseNet201**

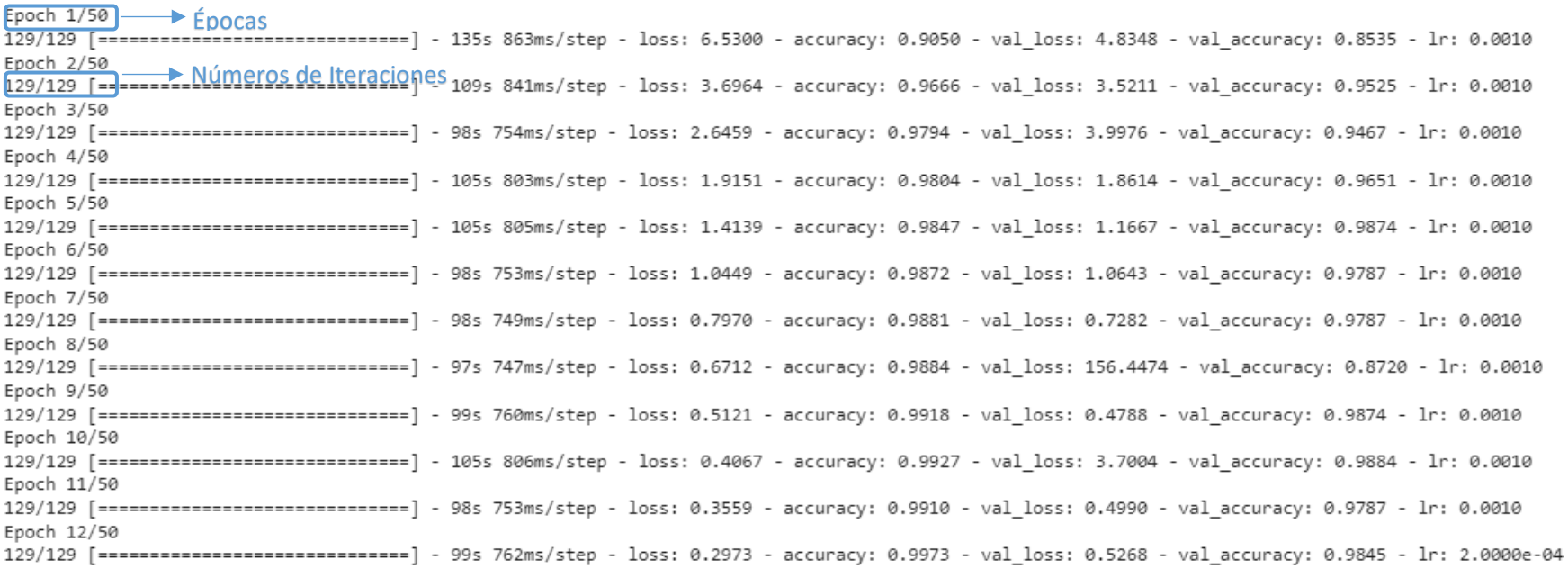



Figura 46. *Iteraciones de Aprendizaje, Modelo DenseNet201*

## B. Ciclo de entrenamiento del Modelo MobileNetV2

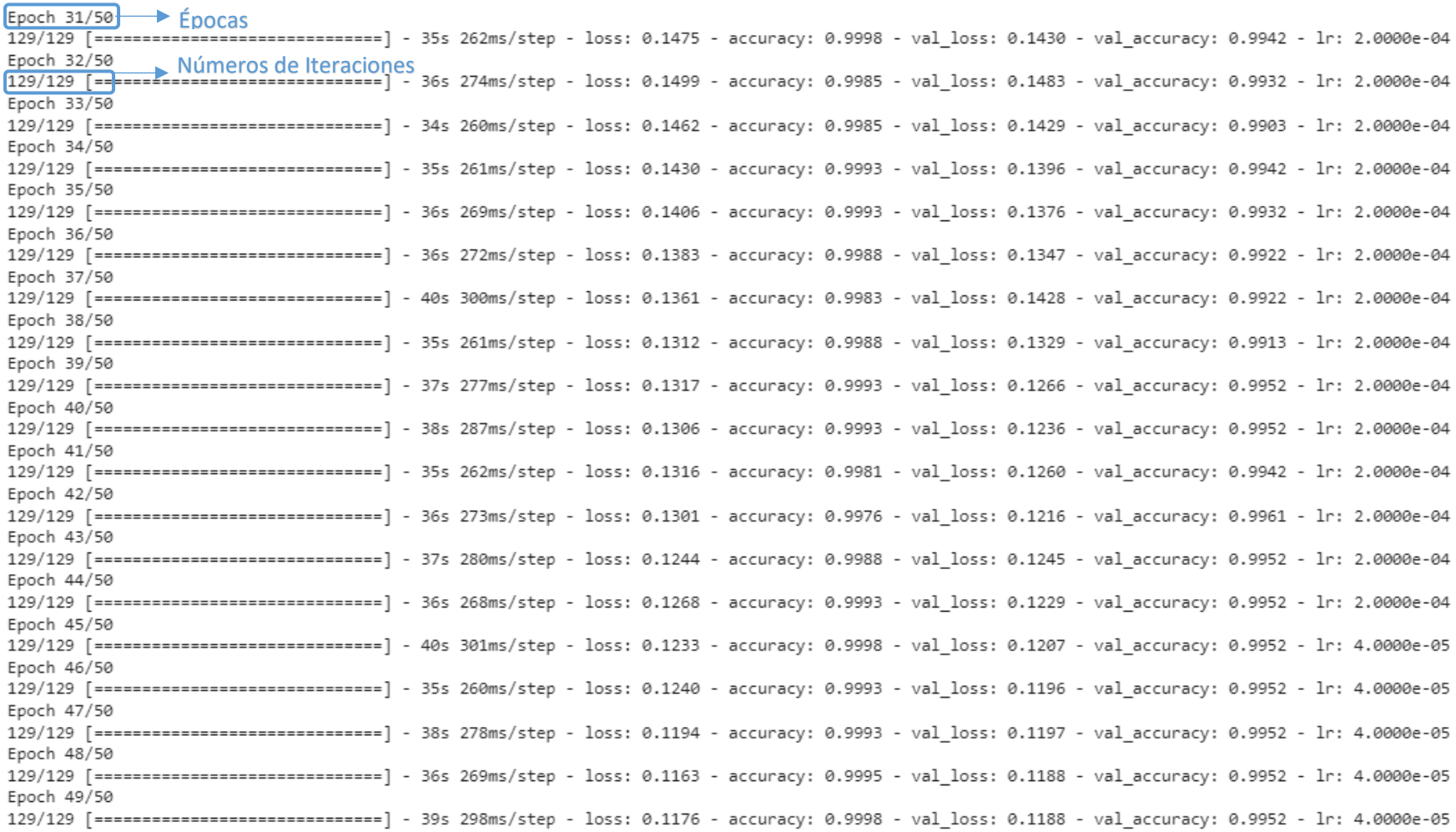



Figura 47. *Iteraciones de Aprendizaje, Modelo MobileNetV2*

## C. Ciclo de entrenamiento del Modelo ResNet50

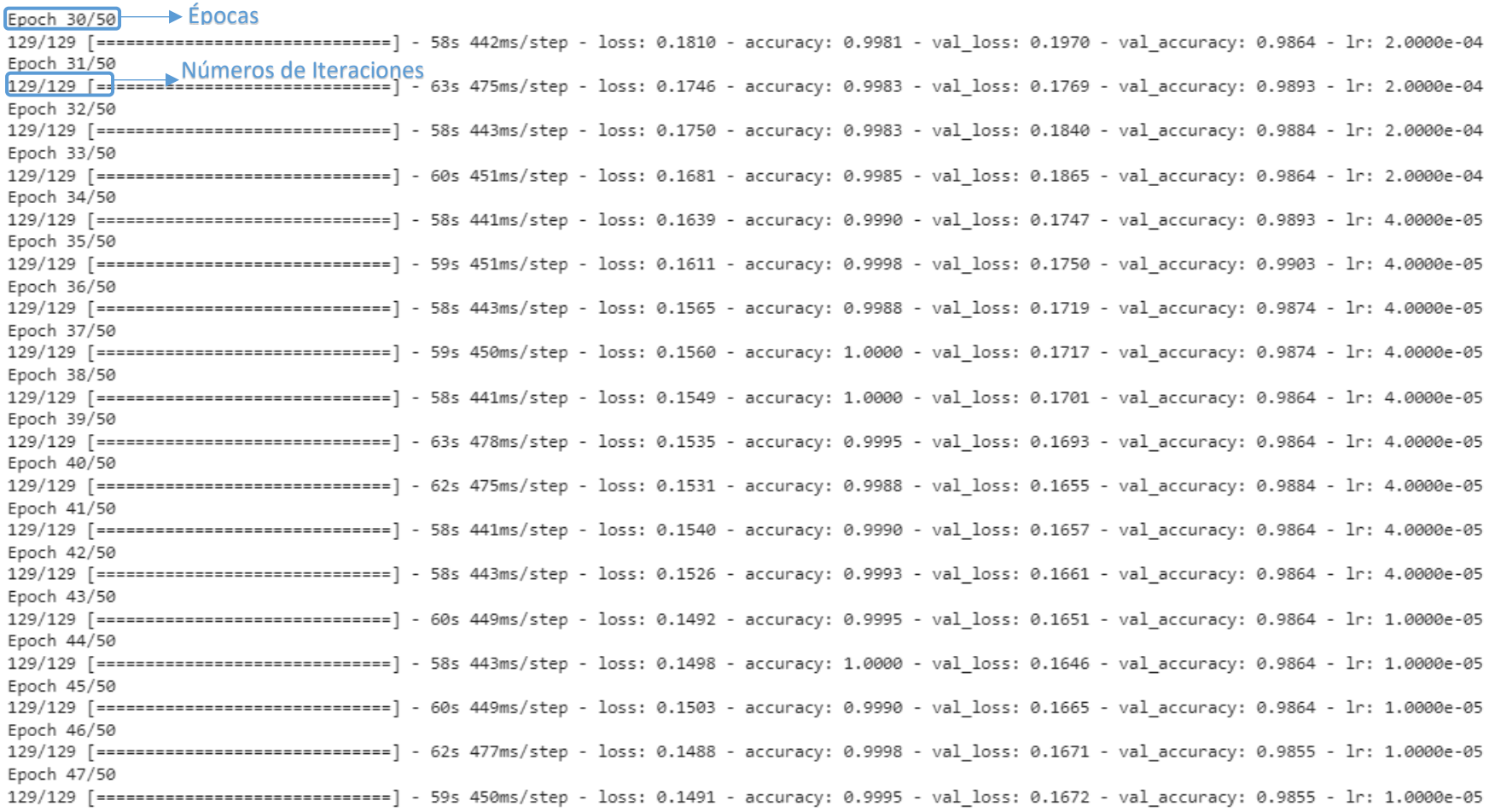
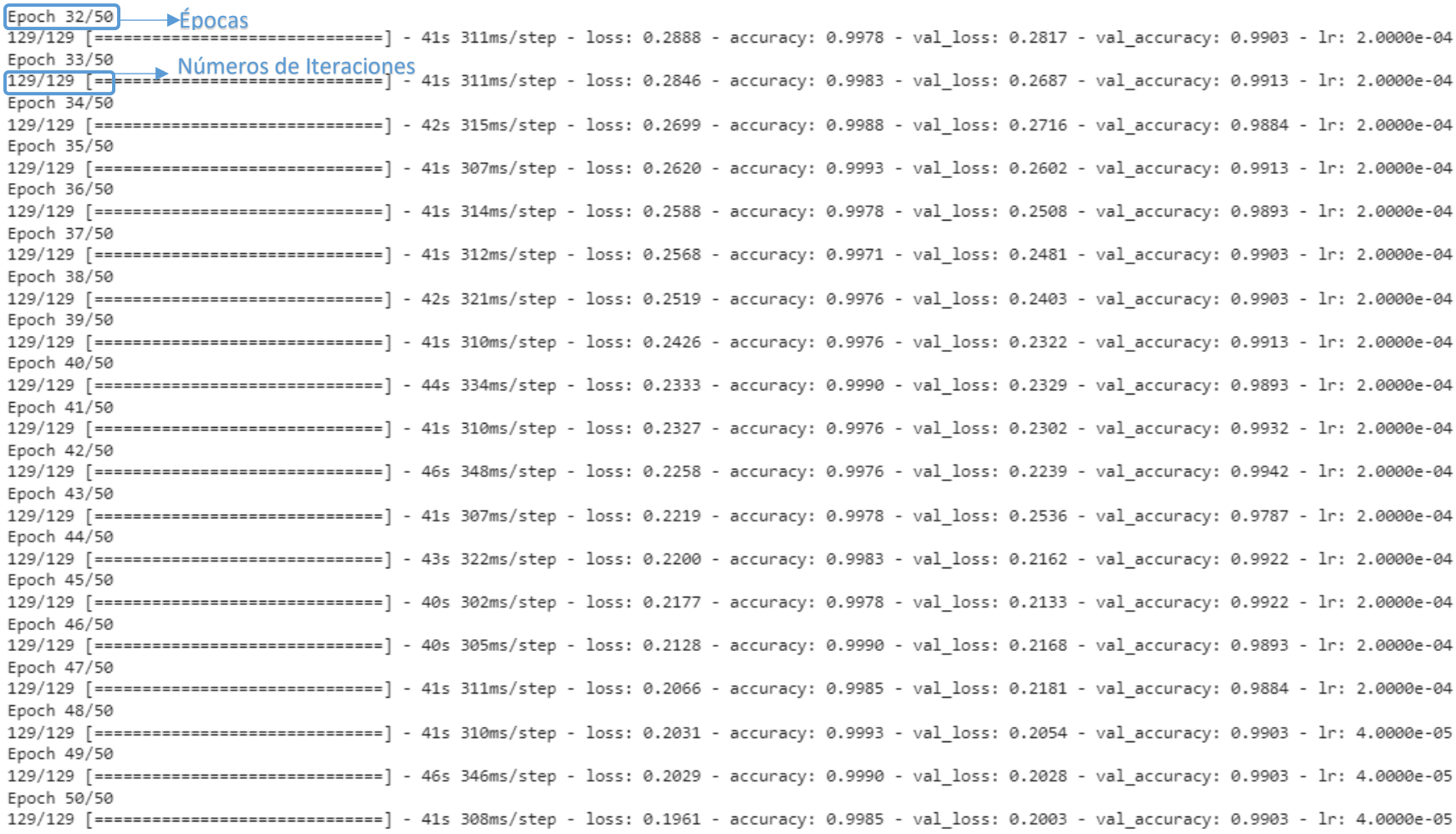



Figura 48. *Iteraciones de Aprendizaje, Modelo ResNet50*

## D. Ciclo de entrenamiento del Modelo EfficientNetV2BO




```
Epoch 32/50
129/129 [==============================] - 41s 311ms/step - loss: 0.2888 - accuracy: 0.9978 - val_loss: 0.2817 - val_accuracy: 0.9903 - lr: 2.0000e-04
Epoch 33/50
129/129 [==============================] - 41s 311ms/step - loss: 0.2846 - accuracy: 0.9983 - val_loss: 0.2687 - val_accuracy: 0.9913 - lr: 2.0000e-04
Epoch 34/50
129/129 [==============================] - 42s 315ms/step - loss: 0.2699 - accuracy: 0.9988 - val_loss: 0.2716 - val_accuracy: 0.9884 - lr: 2.0000e-04
Epoch 35/50
129/129 [==============================] - 41s 307ms/step - loss: 0.2620 - accuracy: 0.9993 - val_loss: 0.2602 - val_accuracy: 0.9913 - lr: 2.0000e-04
Epoch 36/50
129/129 [==============================] - 41s 314ms/step - loss: 0.2588 - accuracy: 0.9978 - val_loss: 0.2508 - val_accuracy: 0.9893 - lr: 2.0000e-04
Epoch 37/50
129/129 [==============================] - 41s 312ms/step - loss: 0.2568 - accuracy: 0.9971 - val_loss: 0.2481 - val_accuracy: 0.9903 - lr: 2.0000e-04
Epoch 38/50
129/129 [==============================] - 42s 321ms/step - loss: 0.2519 - accuracy: 0.9976 - val_loss: 0.2403 - val_accuracy: 0.9903 - lr: 2.0000e-04
Epoch 39/50
129/129 [==============================] - 41s 310ms/step - loss: 0.2426 - accuracy: 0.9976 - val_loss: 0.2322 - val_accuracy: 0.9913 - lr: 2.0000e-04
Epoch 40/50
129/129 [==============================] - 44s 334ms/step - loss: 0.2333 - accuracy: 0.9990 - val_loss: 0.2329 - val_accuracy: 0.9893 - lr: 2.0000e-04
Epoch 41/50
129/129 [==============================] - 41s 310ms/step - loss: 0.2327 - accuracy: 0.9976 - val_loss: 0.2302 - val_accuracy: 0.9932 - lr: 2.0000e-04
Epoch 42/50
129/129 [==============================] - 46s 348ms/step - loss: 0.2258 - accuracy: 0.9976 - val_loss: 0.2239 - val_accuracy: 0.9942 - lr: 2.0000e-04
Epoch 43/50
129/129 [==============================] - 41s 307ms/step - loss: 0.2219 - accuracy: 0.9978 - val_loss: 0.2536 - val_accuracy: 0.9787 - lr: 2.0000e-04
Epoch 44/50
129/129 [==============================] - 43s 322ms/step - loss: 0.2200 - accuracy: 0.9983 - val_loss: 0.2162 - val_accuracy: 0.9922 - lr: 2.0000e-04
Epoch 45/50
129/129 [==============================] - 40s 302ms/step - loss: 0.2177 - accuracy: 0.9978 - val_loss: 0.2133 - val_accuracy: 0.9922 - lr: 2.0000e-04
Epoch 46/50
129/129 [==============================] - 40s 305ms/step - loss: 0.2128 - accuracy: 0.9990 - val_loss: 0.2168 - val_accuracy: 0.9893 - lr: 2.0000e-04
Epoch 47/50
129/129 [==============================] - 41s 311ms/step - loss: 0.2066 - accuracy: 0.9985 - val_loss: 0.2181 - val_accuracy: 0.9884 - lr: 2.0000e-04
Epoch 48/50
129/129 [==============================] - 41s 310ms/step - loss: 0.2031 - accuracy: 0.9993 - val_loss: 0.2054 - val_accuracy: 0.9903 - lr: 4.0000e-05
Epoch 49/50
129/129 [==============================] - 46s 346ms/step - loss: 0.2029 - accuracy: 0.9990 - val_loss: 0.2028 - val_accuracy: 0.9903 - lr: 4.0000e-05
Epoch 50/50
129/129 [==============================] - 41s 308ms/step - loss: 0.1961 - accuracy: 0.9985 - val_loss: 0.2003 - val_accuracy: 0.9903 - lr: 4.0000e-05
```

Figura 49. *Iteraciones de Aprendizaje, Modelo EfficienteNetV2BO*

La siguiente tabla especifica el porcentaje (80%) de imágenes tomadas para el entrenamiento, la cantidad de imágenes (32), que se tomaran para cada pasada(epoch). La cantidad de epoch

| % img | Bach_size | Iteration |
|---|---|---|
| 80% | 32 | 1 |
| | 32 | 2 |
| | : | : |
| | 32 | 128 |
| | 32 | 129 |
| 4127 ÷ 32 = 129 | | 1 epoch |

*Tabla 10. Detalles de entrenamiento*

### Guardado del Histórico del entrenamiento

```
import pickle
import datetime
log_file_path = "/content/drive/MyDrive/project/logs/DenseNet201-"
+ datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
with open(log_file_path, 'wb') as file_pi:
    pickle.dump(history.history, file_pi)
```

Figura 50. *Guardado del histórico de entrenamiento*

### Cambios Efectuados en cada Notebook de los Modelos

| Cambio | Líneas de Código afectadas para cada modelo | Extensión |
|---|---|---|
| **Para cada Archivo de los Modelos se actualizo el nombre del mismo** | `# Pretrained model`<br>`base_model = keras.applications.ResNet50(`<br>`include_top=False,`<br>`weights="imagenet",`<br>`input_shape=input_shape)` | .ipynb |
| | `tf.keras.callbacks.ModelCheckpoint(filepath=os.path.join (config["save_model"]["path"], "ResNet50.{epoch:02d}-{val_accuracy:.2f}.h5"),` | |
| | `log_file_path = "/content/drive/MyDrive/project/logs/ResNet50-" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")` | |

*Tabla 11. Configuración de cada Modelo*

### 4.1.4. Predicción

En la predicción lo que primero se hace es importar una función **%pylab().** La que va a permitir traer consigo varios módulos de **Matplotlib** y que se van a usar en la importación de las librerías para visualizar la identificación de la hoja enferma.

```
%pylab inline
img_path =
"/content/drive/MyDrive/project/data/unzipped/data/Armyworm/20230310_114951.JPG"
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread(img_path)
imgplot = plt.imshow(img)
plt.show();
img = keras.preprocessing.image.load_img(img_path, target_size=image_size)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
score = predictions[0]
print(
    f"This image is:\n {score[0]:.2f}
    Armyworm\n {score[1]:.2f}
    percent Blight\n {score[2]:.2f}
    percent Common_Rust\n {score[3]:.2f}
    Healthy")
```

Figura 51. *Predicción de Enfermedad*

Primero se lee la ruta de la imagen, seguido se va cargar la imagen con la función de **load_img()** en formato PIL. Donde tiene como argumentos a la ruta y el tamaño de la imagen. Ya con la imagen cargada se procede a convertir toda la estructura de la imagen en un arreglo **numpy**. Biblioteca que trabajamos con **python** y **keras**.

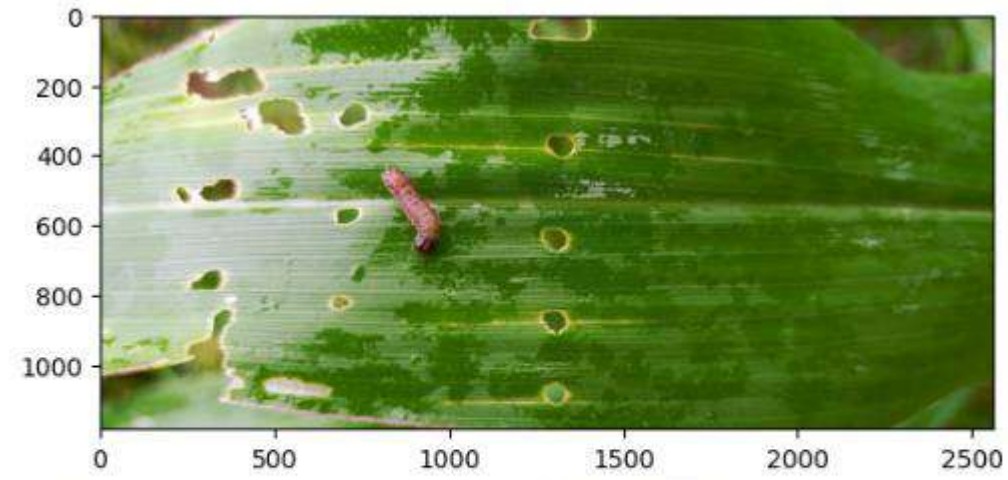
Luego ese conjunto de números almacenados en la variable **img_array** la vamos a representar en un plano. El mismo que se elabora mediante la función **tf.expand_dims().** En el que recibe como parámetros el

arreglo **numpy** y el **axis=0** que representa el valor donde inicializa la representación.

Finalmente se aplica la función **model.predict()** para predecir el resultado de las entradas. Y luego se imprime la imagen con el valor predicho más la etiqueta de la enfermedad.

### Predicción de Plaga por cada Modelo Entrenado

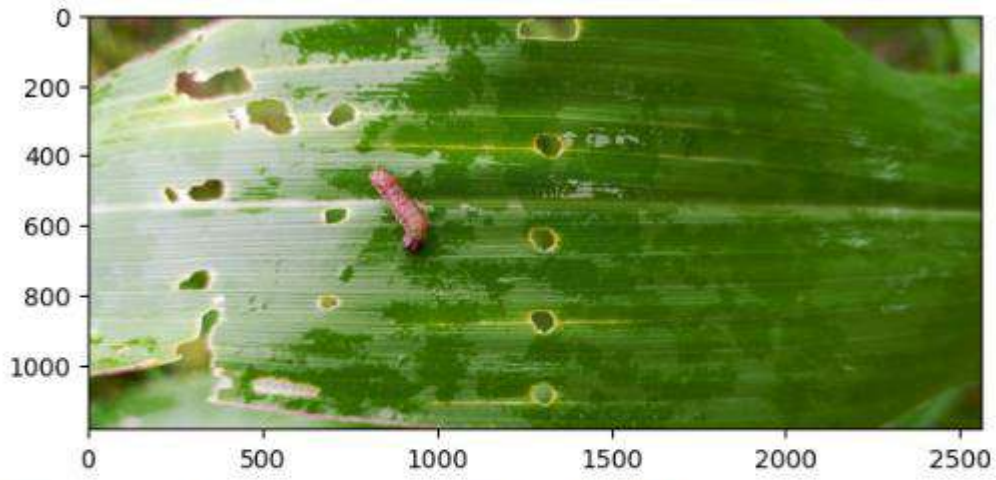
#### A. Predicción Modelo DenseNet201




```
1/1 [==============================] - 0s 163ms/step
This image is:
 0.99 Armyworm
 0.00 percent Blight
 0.00 percent Common_Rust
 0.00 Healthy
```

Figura 52. *Predicción Modelo DenseNet201*

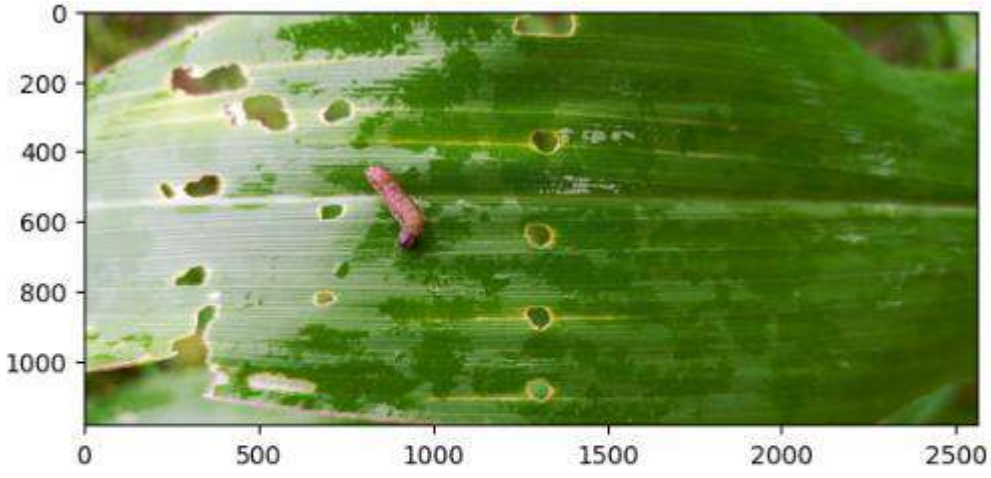
### B. Predicción Modelo MobileNetV2




```
1/1 [==============================] - 0s 23ms/step
This image is:
 1.00 Armyworm
 0.00 percent Blight
 0.00 percent Common_Rust
 0.00 Healthy
```

Figura 53. *Predicción Modelo MobileNetV2*

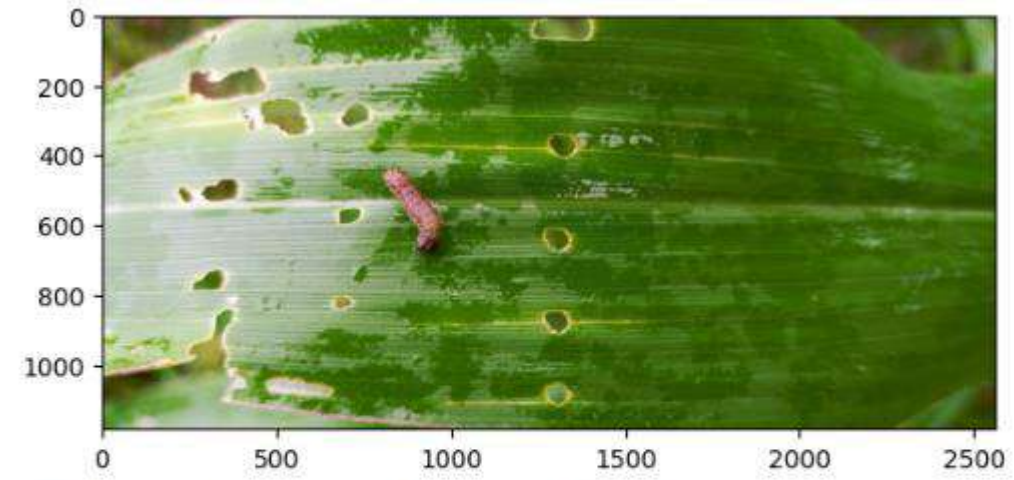
### C. Predicción Modelo ResNet50




```
1/1 [==============================] - 3s 3s/step
This image is:
 0.99 Armyworm
 0.01 percent Blight
 0.00 percent Common_Rust
 0.00 Healthy
```

Figura 54. *Predicción Modelo ResNet50*

**D. Predicción Modelo EfficientNetV2B0**




```
1/1 [==============================] - 2s 2s/step
This image is:
 0.99 Armyworm
 0.00 percent Blight
 0.00 percent Common_Rust
 0.00 Healthy
```

Figura 55. *Predicción de Modelo EfficientNetV2B0*

4.1.5. Interpretación

4.1.4.1. Ingesta de Logs

Para la Interpretación de resultados se importa las siguientes librerías usadas en Python.

```
import pandas as pd
import plotly.express as px
import pickle
import os
import tensorflow as tf
```

Figura 56. *Librerías importadas para Interpretación*

Montamos el enlace del Drive para luego ser abierto del nuestro proyecto que se encuentran en el Drive de google.

```
from google.colab import drive
drive.mount('/content/drive')
```

Figura 57. *Enlace a Drive*

Se establece la ruta y listamos todos los resultados obtenidos del entrenamiento de cada modelo en la carpeta logs.

```
path_logs = "/content/drive/MyDrive/project/logs"
logs = os.listdir(path_logs)
full_logs = [os.path.join(path_logs, log) for log in logs]
```

Figura 58. *Pasar resultados de entrenamiento a variable*

Cargamos resultados de las predicciones de todos los modelos mediante un DataFrame, utilizando funciones integradas con las importaciones antes mencionadas en este paso.

```
data_logs = []
for full_log, log in zip(full_logs, logs):
  with open(full_log, "rb") as file_pi:
    history = pd.DataFrame(pickle.load(file_pi))
    history["model"] = log
    data_logs.append(history)

data_logs = pd.concat(data_logs)
data_logs
```

Figura 59. *Recorrer resultados de todos los modelos*

Lista el resultado del entrenamiento de cada epoch de todos los modelos. Estos valores son las métricas que nos permiten estimar si cada modelo seleccionado se ajusta a los datos de entrenamiento y validación. Tenemos:

**Loss:** nos muestra los valores de pérdida durante el entrenamiento y que son menores que los de val_loss.

**Val_loss:** nos muestra los valores pérdida de validación y en los cuales nos debemos centrar. Porque para saber que nuestro modelo va aprendiendo de manera correcta,y determinar que **no** tiene sobreajuste y generaliza bien. Este valor debería de llegar a un punto donde los valores casi sean iguales o los de **loss.** Y por lo que podemos observar nuestros modelos van por buen camino. Lo mismo que se interpreta como un buen aprendizaje.

**Accuracy:** nos arroja el valor de precisión de los datos de entrenamiento que se encuentra en la siguiente ruta: **/content/drive/MyDrive/project/data/unzipped/data/**

**Val_accuracy:** es el resultado de la precisión del conjunto de datos de validación por cada epoch. Normalmente el valor de esta variable es menor que *accuracy* pero su diferencia no debería ser muy significativa. En nuestros modelos las diferencias son mínimas.

**Lr:** La variable lr (learning rate) es dependiente del parámetro val_loss. Donde si el val_loss no disminuya, el lr baje más para que el modelo de pasos más cortos para llegar al objetivo esperado, es decir a una mejor predicción en val_accuracy. y este valor en cada iteración disminuye más, Mientras val_loss no disminuya.

DataFrame de Predicciones por Modelo. Se muestra todos los epochs de cada modelo.

| | loss | accuracy | val_loss | val_accuracy | lr | model |
|---|---|---|---|---|---|---|
| **0** | 6.529956 | 0.905016 | 4.834826 | 0.853540 | 0.00100 | DenseNet201-20230610-043821 |
| **1** | 3.696426 | 0.966562 | 3.521076 | 0.952473 | 0.00100 | DenseNet201-20230610-043821 |
| **2** | 2.645936 | 0.979404 | 3.997584 | 0.946654 | 0.00100 | DenseNet201-20230610-043821 |
| **3** | 1.915097 | 0.980373 | 1.861350 | 0.965082 | 0.00100 | DenseNet201-20230610-043821 |
| **4** | 1.413942 | 0.984735 | 1.166743 | 0.987391 | 0.00100 | DenseNet201-20230610-043821 |
| **...** | ... | ... | ... | ... | ... | ... |
| **45** | 0.124032 | 0.999273 | 0.119646 | 0.995150 | 0.00004 | MobileNetV220230611-031218 |
| **46** | 0.119363 | 0.999273 | 0.119688 | 0.995150 | 0.00004 | MobileNetV220230611-031218 |
| **47** | 0.116267 | 0.999515 | 0.118772 | 0.995150 | 0.00004 | MobileNetV220230611-031218 |
| **48** | 0.117596 | 0.999758 | 0.118841 | 0.995150 | 0.00004 | MobileNetV220230611-031218 |
| **49** | 0.119166 | 0.999273 | 0.118516 | 0.994180 | 0.00004 | MobileNetV220230611-031218 |

159 rows × 6 columns

Figura 60. *Resultados de entrenamiento*

4.1.4.2. Revisión de los Entrenamientos

- ✓ **Precisión Máxima**

Para comparar la precisión de los modelos lo que se hizo fue usar las librerías pandas y agrupar por modelo. Comparando el val_accuracy y el número de epoch en el que ocurrió ese valor.

Con los numero obtenidos deducimos que el modelo DenseNet201 fue el modelo óptimo debido a que tan solo 10 epochs le hicieron suficientes para lograr una precisión de 0.988361. Luego le sigue ResNet50 con un valor de 0.990301 en 13 epochs que fue mucho mejor pero que le tomo mayor número de épocas(pasadas), seguido por MobileNetV2 con un resultado de 0.996120 con 42 epochs y finalmente EfficienteNetV2BO 0.994180 en el epochs 42.

Es la máxima precisión obtenida por cada modelo y las epochs que le tomó lograrlo: DenseNet201 logra la mayor precisión en el menor número de epochs, pero es el modelo más pesado; aunque no el que tiene el mayor número de parámetros.

```
effort = data_logs.groupby("model")["val_accuracy"].agg(["max", "idxmax"])
effort["epoch"] = effort["idxmax"] + 1
effort.drop("idxmax", axis=1, inplace=True)
effort
```

| model | max | epoch |
|---|---|---|
| DenseNet201-20230610-043821 | 0.988361 | 10 |
| EfficienteNetV2BO-20230610-061923 | 0.994180 | 42 |
| MobileNetV220230611-031218 | 0.996120 | 42 |
| ResNet50-20230610-074818 | 0.990301 | 13 |

Figura 61. *Mejores resultados obtenidos por modelo*

- ✓ **Peso o Costo de Entrenamiento de los Modelos**

Estos son los pesos y números de parámetros de cada modelo. Lo que cuesta entrenarlos.

| | size_MB | parameters_M |
|---|---|---|
| **ResNet50** | 50.0 | 25.6 |
| **EfficienteNetV2BO** | 29.0 | 7.2 |
| **MobileNetV2** | 14.0 | 3.5 |
| **DenseNet201** | 80.0 | 20.2 |

Figura 62. *Costo de entrenamiento de los modelos*

4.1.4.3. Gráficos de Curvas de Entrenamiento.

1. Resultados de Pérdida.

En esta parte se visualizar el resultado de las métricas loss y val_loss. Los mismos que su tendencia era disminuir con un margen de diferenciación mínimo en cada modelo. Como el margen de pérdida no fue en aumento y los mínimos tuvieron resultados casi estables, se interpreta como un entrenamiento adecuado.

```
data_logs["epoch"] = data_logs.index + 1
fig = px.line(data_logs,
              x="epoch",
              y=["val_loss"],
              color="model",
              template="plotly_white",
              title="val_loss")
fig.show()
```

Figura 63. *Configuración de argumentos para grafico de pérdida*

En el gráfico siguiente se muestran los resultados del modelo **DenseNet201.** Resultados obtenidos durante el entrenamiento, con un valor en validación de pérdida de val_loss = 3.7004, en la epoch 10. En el eje Y el valor de pérdida de validación y en el eje X el número de épocas que son 50 durante el entrenamiento de cada modelo. El número de épocas dependerá del valor de la mejor precisión. También se visualiza una leyenda que representa cada modelo con un color definido. Con esta gráfica lo que se interpreta es que la tendencia del valor de pérdida de validación es a disminuir.

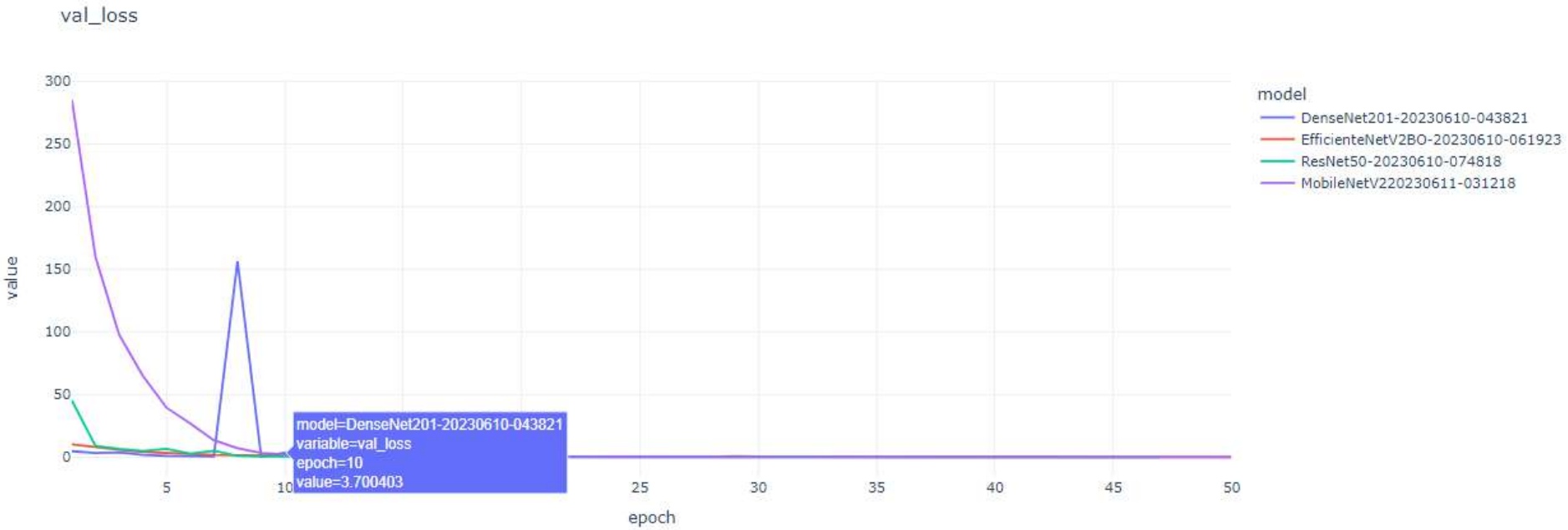



Figura 64. *Nivel de pérdida del Modelo DenseNet201*

2. Resultados de Precisión

```
fig = px.line(data_logs, x="epoch", y=["val_accuracy"], color="model",
              template="plotly_white", title="val_accuracy")
fig.update_layout(title_x=0.5)
fig.show()
```

Figura 65. *Configuración de argumentos para gráfico predicción*

En el siguiente gráfico lo que se representa es la precisión de la validación del modelo DenseNet201 val_accuracy = 0.9884 con tan solo 10 épocas(epoch)

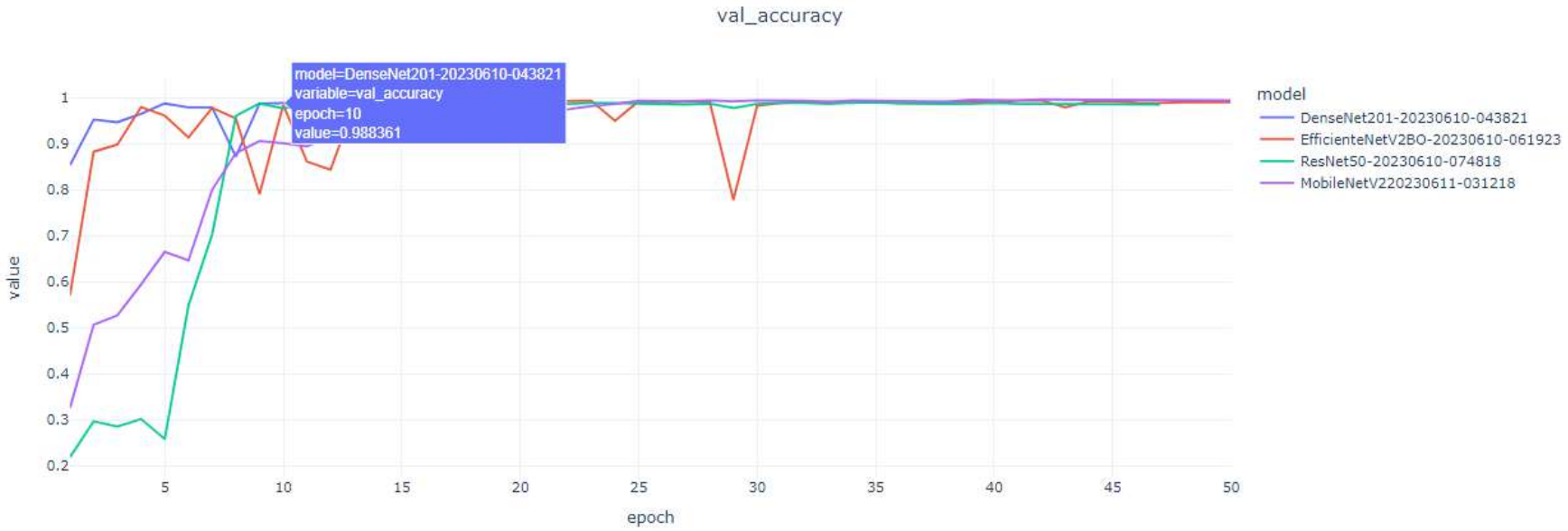



Figura 66. *Nivel de precisión de modelo DenseNet201*

4.1.4.4. Predicciones

a) Importar librerías y Cargar el mejor modelo.

En esta parte se carga aquel modelo que obtuvo mejores resultados durante el entrenamiento. Considerando como aspecto importante el tiempo de procesamiento. Le llego a tomar solo 10 epochs para obtener una predicción del 0.988361. Es decir, cerca del 99% de acierto. Ese modelo es **DenseNet201.**

```
import keras
import tensorflow
model = keras.models.load_model
("/content/drive/MyDrive/project/models/DenseNet201.10-0.99.h5")
```

Figura 67. *Librerías usadas para carga de predicción*

b) Carga de Imágenes desconocidas por el modelo y Realizamos Predicciones sobre ellas.

```
%pylab inline
images = ["/content/drive/MyDrive/project/data/test2/Armyworm.jpg",
          "/content/drive/MyDrive/project/data/test2/common_rust.jpg",
           "/content/drive/MyDrive/project/data/test2/blight.jpeg",
          "/content/drive/MyDrive/project/data/test2/Healthy.jpeg"]

for image in images:
  img_path = image
  import matplotlib.pyplot as plt
  import matplotlib.image as mpimg
  img = mpimg.imread(img_path)
  imgplot = plt.imshow(img)
  plt.show();
  img = tensorflow.keras.utils.load_img(img_path, target_size=(240, 240))
  img_array = tensorflow.keras.utils.img_to_array(img)
  img_array = tensorflow.expand_dims(img_array, 0)  # Create batch axis

  predictions = model.predict(img_array)
  score = predictions[0]
```

Figura 68. *Carga de imágenes desconocidas por el modelo*

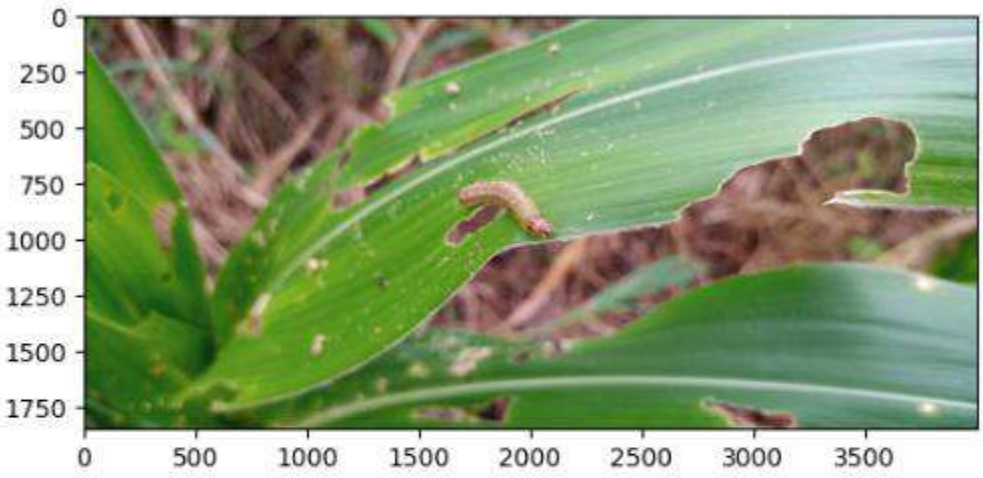
En esta parte lo que se realiza es cargar las imágenes nuevas, construir un ciclo **for image in images:** con el objetivo de iterar cada imagen en un plano. Para poder realizar la predicción y determinar la precisión de cada imagen procesada.

En las siguientes líneas de código lo que se hace es mostrar cada imagen con su nombre de archivo, descripción de las enfermedades y el valor referente a la precisión determinada por el modelo.

```
print(
    f"This image is:\n {score[0]:.2f}
    Armyworm\n {score[1]:.2f}
    percent Blight\n {score[2]:.2f}
    percent Common_Rust\n {score[3]:.2f}
    Healthy")
```

Figura 69. *Mostrar imagen etiquetada*

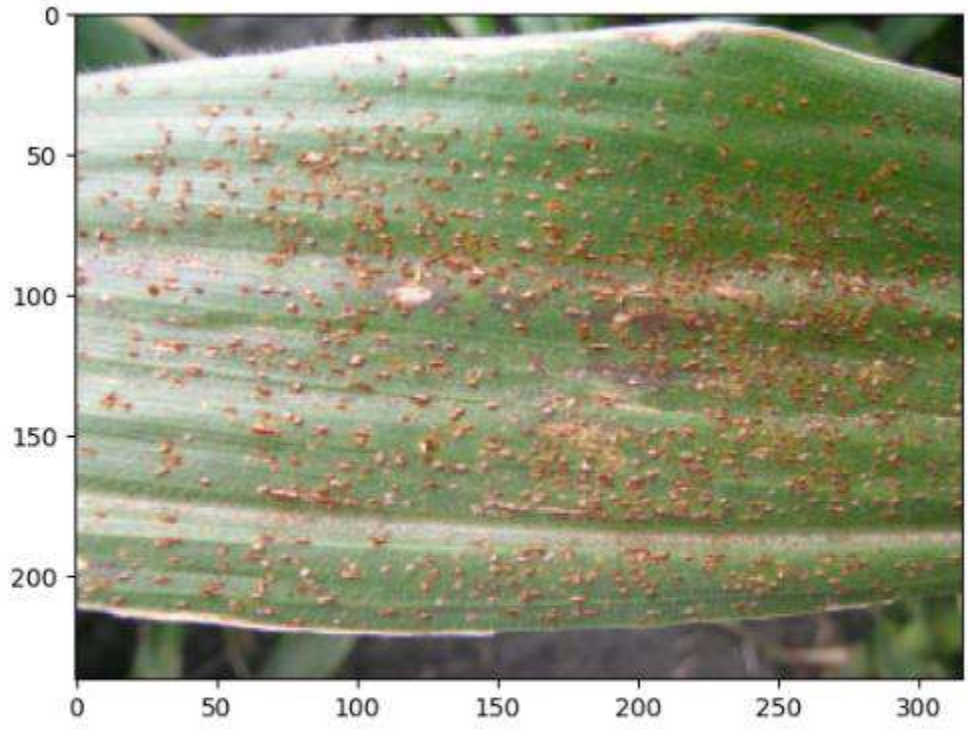
En la primera predicción, el modelo determina que la imagen se clasifica en el grupo de Plagas como Gusano. Con una precisión de 0.97, es decir al 97% acierta que detecto la plaga que está atacando es la mencionada.




```
1/1 [==============================] - 13s 13s/step
This image is:
 0.97 Armyworm
 0.01 percent Blight
 0.01 percent Common_Rust
 0.01 Healthy
```

Figura 70. *Predicción de Plaga Gusano*

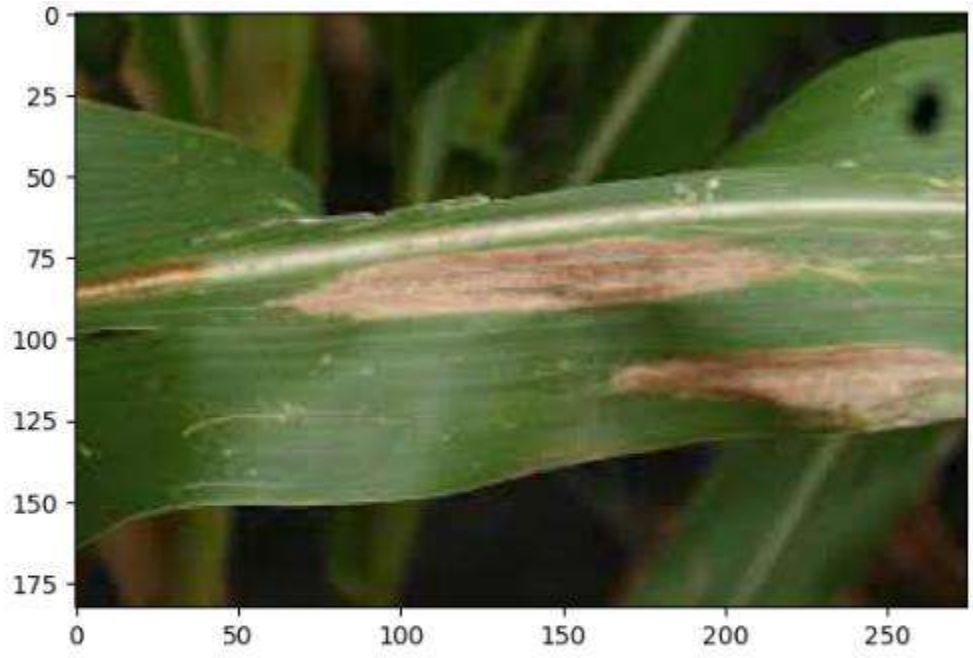
El modelo determina a un 0.95, es decir a un 95 % que la imagen de la enfermedad se encuentra dentro del conjunto de *Roya Común*. Este valor nos arroja una precisión adecuada para la identificación.




```
1/1 [==============================] - 0s 81ms/step
This image is:
 0.02 Armyworm
 0.02 percent Blight
 0.95 percent Common_Rust
 0.01 Healthy
```

Figura 71. *Predicción de enfermedad mancha foliar*

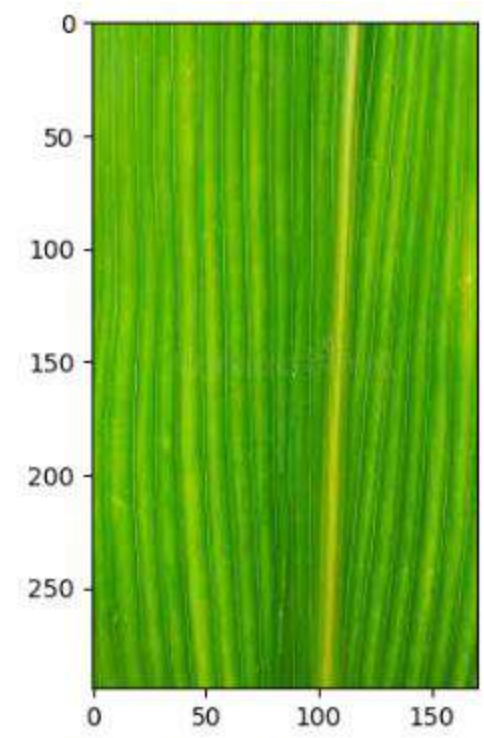
Una precisión de 0.98, Nos dice que la enfermedad detectada es *Tizón de la Hoja de Maíz*. Al 98% es un valor muy eficiente con respecto a la identificación.




```
1/1 [==============================] - 0s 93ms/step
This image is:
 0.01 Armyworm
 0.98 percent Blight
 0.01 percent Common_Rust
 0.00 Healthy
```

Figura 72. *Predicción de enfermedad Tizón de la hoja de maíz*

La última predicción determina al 0.73 que es una hoja de maíz saludable con 73% de acierto.




```
1/1 [==============================] - 0s 37ms/step
This image is:
 0.10 Armyworm
 0.09 percent Blight
 0.07 percent Common_Rust
 0.73 Healthy
```

Figura 73. *Predicción de hoja saludable*

## V. DISCUSIÓN DE RESULTADOS

En este capítulo evaluaremos el rendimiento de los modelos para dar respuesta a la hipótesis formulada en el proyecto.

### 5.1. Formulación del Problema

¿Cómo mejorar la detección temprana de plagas y enfermedades en los cultivos de maíz en el Distrito de Cascas, La Libertad?

### 5.2. Formulación de Hipótesis

Un modelo de procesamiento de imágenes con Redes Neuronales detecta plagas y enfermedades en los cultivos de maíz en el Distrito de Cascas, La Libertad, durante el año 2022.

### 5.3. Validación de Resultados

Primero se importan las librerías necesarias a utilizar para el graficar la matriz de confusión.

```
from sklearn.metrics import confusion_matrix, f1_score, roc_curve,
precision_score, recall_score, accuracy_score, roc_auc_score
from sklearn import metrics

from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Figura 74. *Importación de Librerías*

Luego se almacenan las etiquetas de cada categoría en la variable names mediante un arreglo. Seguido se pasa una función de lectura de datos a otra variable. La misma que se va a usar con otra función keras flow_from_directory que permitir dividir las imágenes en lotes, donde se tiene configurador algunos parámetros. Y también donde se encuentra la ruta de acceso a las imágenes nuevas para medir el rendimiento de cada modelo, mediante la matriz de confusión. Para dicha tarea se consideró un total de 320 imágenes. Las mismas que están comprendidas en grupos de 80 imágenes cada carpeta.

```
names = ['Armyworm','Blight','Common_Rust','Healthy']
```

```
#ImageDataGenerator() para leer los datos de test
test_datagen = ImageDataGenerator()

test_generator = test_datagen.flow_from_directory(
    config["data_test"]["test_images"],
    target_size=(240,240),
    batch_size=batch_size,
    class_mode='categorical',
#se configura en false para garantizar que las predicciones
#se ordenen y coincidan con las etiquetas
    shuffle=False
)
```

```
Found 320 images belonging to 4 classes.
```

Figura 75. *Función para agrupación por lotes*

El siguiente paso carga el modelo con la mejor la información obtenida durante el entrenamiento y que ahora se usarán para predecir. Para ello se pasa las variables configuradas con anterioridad a una nueva función predict_generator. Que va a permitir predecir los resultados de las imágenes que se le pase. Adicional a ello se crean dos variables donde va a recibir la información de la data predicha y la real.

```
#custom_Model= load_model("DenseNet201.10-0.99.h5")
custom_Model = load_model("/content/drive/MyDrive/project/models/DenseNet201.10-0.99.h5")

predictions = custom_Model.predict_generator(generator=test_generator)

y_pred = np.argmax(predictions, axis=1)
y_real = test_generator.classes
```

Figura 76. *Predicción de imágenes*

Finalmente se utiliza la función de la matriz de confusión confusión_matrix para evaluar las métricas de cada modelo entrenado. Donde se van a representar mediante una tabla de filas y columnas con la librería plot. Pasandole algunos parámetros.

```
cm=metrics.confusion_matrix(y_real, y_pred)

plot_confusion_matrix(conf_mat=cm, figsize=(6,6), class_names = names, show_normed=False)
plt.tight_layout()
```

Figura 77. *Generación de la matriz de Confusión*

Se sabe que se tiene 4 clases:

| Nombre en Inglés | Nombre Técnico |
|---|---|
| **Armyworm** | Gusano |
| **Blight** | Tizón de la Hoja |
| **Common Rust** | Roya Común |
| **Healthy** | Saludables |

*Tabla 12. Plaga y Enfermedades del Maíz*

A continuación de describe la estructura y los valores obtenidos de la matriz de confusión de cada uno de los modelos.

Las filas están conformadas por los valores o imágenes reales y las columnas por los valores predichos durante el entrenamiento. La tabla se conforma por la siguiente nomenclatura:

TP = verdaderos positivos

VN = verdaderos negativos

FP = falsos positivos

FN = falsos negativos

| | Data predicha | |
|---|---|---|
| **Data real** | TP | FN |
| | FP | TN |

*Tabla 13. Estructura de matriz de confusión*

5.3.1. Resultados del modelo DenseNet201 en la matriz de confusión.

Los TP de la clase **Armyworm** fueron 78 clasificados como gusano. Donde para la clase **Blight** obtuvo 1 clasificados como gusano, en la clase **common_rust** hubo 0 clasificados como gusano, y finalmente en la clase **Healthy** tiene 1 clasificados como gusano.

Los TP de la clase **Blight** fueron 41 clasificados como Blight. Donde para la clase **Armyworm** obtuvo 36 clasificados como Blight, en la clase **common_rust** hubo 3 clasificados como Blight, y finalmente en la clase **Healthy** tiene 0 clasificados como Blight.

Los TP de la clase Common Rust fueron 26 clasificados como Common Rust. Donde para la clase Armyworm obtuvo 52 clasificados como Common Rust, en la clase Blight hubo 1 clasificados como Common Rust, y finalmente en la clase Healthy tiene 1 clasificados como Common Rust.

Los TP de la clase Healthy fueron 34 clasificados como Healthy. Donde para la clase Armyworm obtuvo 46 clasificados como Healthy, en la

clase Blight hubo 0 clasificados como Healthy, y finalmente en la clase Common Rust tiene 0 clasificados como Healthy.

Para lo TN se procede a sumar todos los valores que no pertenecen a la fila y columna del TP.

$TN_{Armyworm} = 41 + 3 + 0 + 1 + 26 + 1 + 0 + 0 + 34$

$TN_{Blight} = 78 + 0 + 1 + 52 + 26 + 1 + 46 + 0 + 34$

$TN_{Common\_Rust} = 78 + 1 + 1 + 36 + 41 + 0 + 46 + 0 + 34$

$TN_{Healthy} = 78 + 1 + 0 + 36 + 41 + 3 + 52 + 1 + 26$

Para lo FP se procede a sumar todos los valores de la columna **excepto el valor de la celda** a la que pertenece la clase.

$FP_{Armyworm} = 36 + 52 + 46$

$FP_{Blight} = 1 + 1 + 0$

$FP_{Common\_Rust} = 0 + 3 + 0$

$FP_{Healthy} = 1 + 0 + 1$

Para lo FN se procede a sumar todos los valores de la fila. Inverso al caso anterior **excepto el valor de la celda** a la que pertenece la clase.

$FN_{Armyworm} = 1 + 0 + 1$

$FN_{Blight} = 36 + 3 + 0$

$FN_{Common\_Rust} = 52 + 1 + 1$

$FN_{Healthy} = 46 + 0 + 0$

| Matriz de Observación DenseNet201 | | | | |
|---|---|---|---|---|
| **CLASES** | MEDIDAS | | | |
| | TP | TN | FP | FN |
| **Armyworm** | 78 | 106 | 134 | 2 |
| **Blight** | 41 | 238 | 2 | 39 |
| **Common Rust** | 26 | 237 | 3 | 54 |
| **Healthy** | 34 | 238 | 2 | 46 |

*Tabla 14. Matriz de Observación de Resultados DenseNet201*

**Matriz de Confusión DenseNet201**

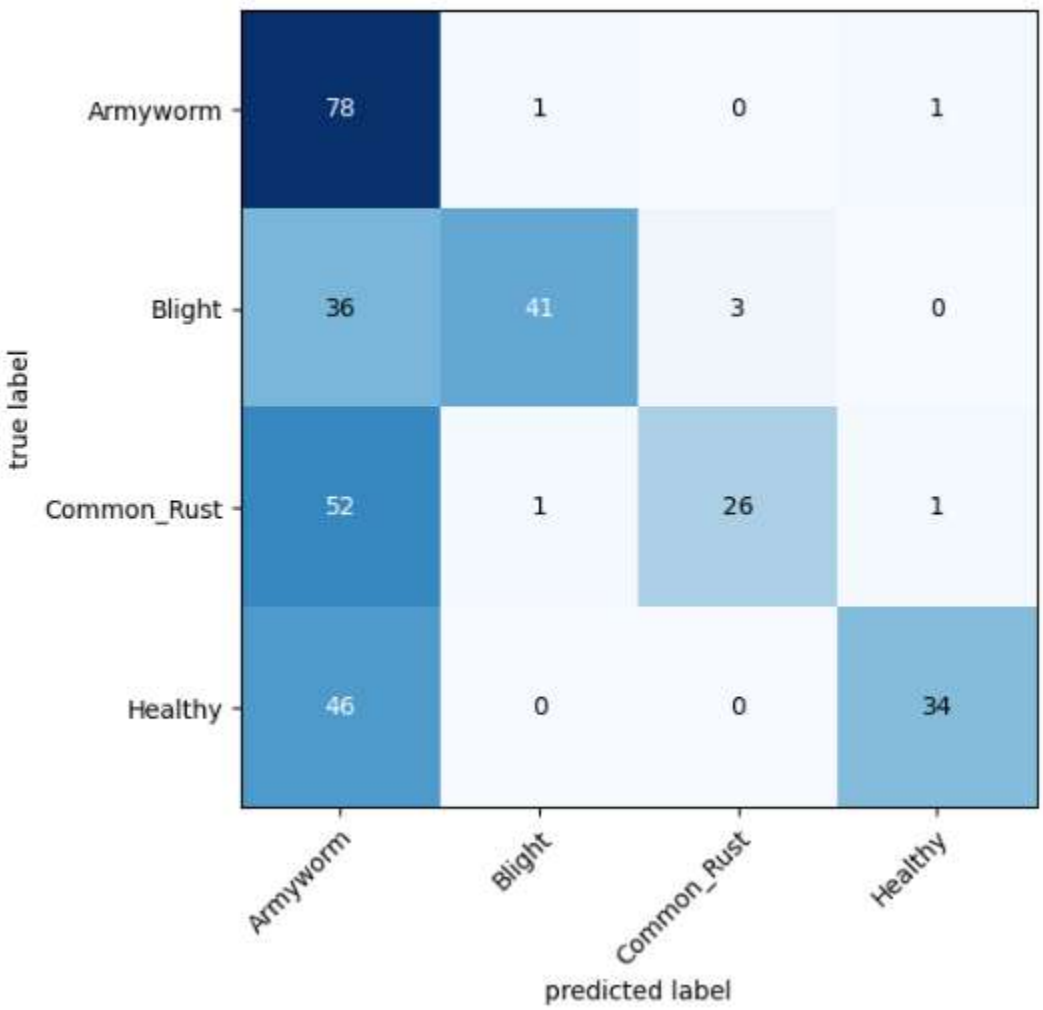



Figura 78. *Matriz de Confusión del Modelo DenseNet201*

5.3.2. Resultados del modelo MobileNetV2 en la matriz de confusión.

Los TP de la clase **Armyworm** fueron 56 clasificados como gusano. Donde para la clase **Blight** obtuvo 10 clasificados como Armyworm, en la clase **common_rust** hubo 0 clasificados como Armyworm, y finalmente en la clase **Healthy** tiene 14 clasificados como Armyworm.

Los TP de la clase **Blight** fueron 55 clasificados como Blight. Donde para la clase **Armyworm** obtuvo 13 clasificados como Blight, en la clase **common_rust** hubo 10 clasificados como Blight, y finalmente en la clase **Healthy** tiene 2 clasificados como Blight.

Los TP de la clase **Common Rust** fueron 59 clasificados como Common Rust. Donde para la clase **Armyworm** obtuvo 16 clasificados como Common Rust, en la clase **Blight** hubo 1 clasificados como Common Rust, y finalmente en la clase **Healthy** tiene 4 clasificados como Common Rust.

Los TP de la clase **Healthy** fueron 20 clasificados como Healthy. Donde para la clase **Armyworm** obtuvo 59 clasificados como Healthy, en la clase **Blight** hubo 1 clasificados como Healthy, y finalmente en la clase **Common Rust** tiene 0 clasificados como Healthy.

Para los TN se procede a sumar todos los valores que no pertenecen a la fila y columna del TP.

TNArmyworm = 55 + 10 + 2 + 1 + 59 + 4 + 1 + 0 + 20

TNBlight = 56 + 0 + 14 + 16 + 59 + 4 + 59 + 0 + 20

TNCommon_Rust = 56 + 10 + 14 + 13 + 55 + 2 + 59 + 1 + 20

TNHealthy = 56 + 10 + 0 + 13 + 55 + 10 + 16 + 1 + 59

Para los FP se procede a sumar todos los valores de la columna **excepto el valor de la celda** a la que pertenece la clase.

FPArmyworm = 13 + 16 + 59

FPBlight = 10 + 1 + 1

FPCommon_Rust = 0 + 10 + 0

FPHealthy = 14 + 2 + 4

Para lo FN se procede a sumar todos los valores de la fila. Inverso al caso anterior excepto el valor de la celda a la que pertenece la clase.

FN Armyworm = 10 + 0 + 14

FN Blight = 13 + 10 + 2

FN Common_Rust = 16 + 1 + 4

FN Healthy = 59 + 1 + 0

| Matriz de Observación MobileNetV2 | | | | |
|---|---|---|---|---|
| **CLASES** | MEDIDAS | | | |
| | TP | TN | FP | FN |
| **Armyworm** | 56 | 152 | 89 | 24 |
| **Blight** | 55 | 228 | 12 | 25 |
| **Common Rust** | 59 | 230 | 10 | 21 |
| **Healthy** | 20 | 220 | 20 | 60 |

*Tabla 15. Matriz de Observación de Resultados MobileNetV2*

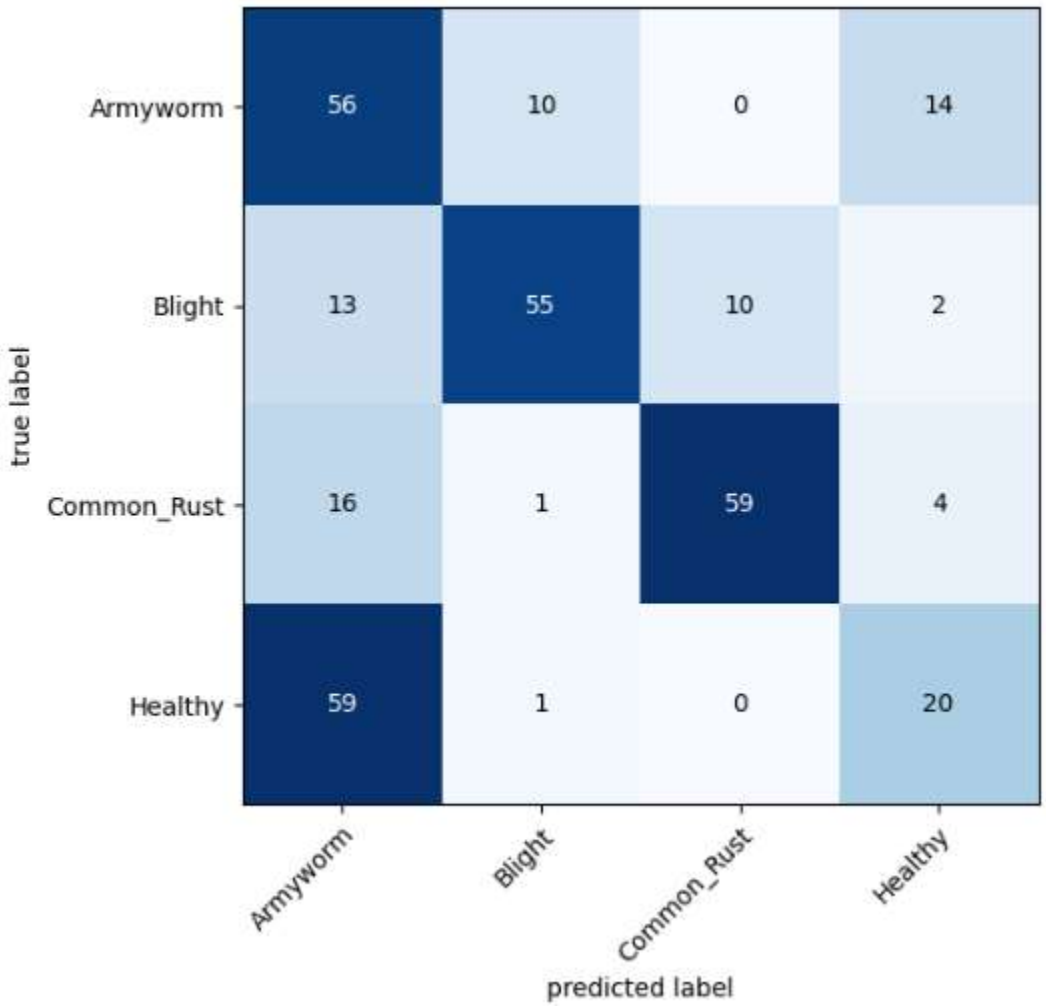



Figura 79. *Matriz de Confusión del Modelo MobileNetV2*

5.3.3. Resultados del modelo EfficientNetV2B0 en la matriz de confusión.

Los TP de la clase **Armyworm** fueron 71 clasificados como gusano. Donde para la clase **Blight** obtuvo 6 clasificados como Armyworm, en la clase **common_rust** hubo 0 clasificados como Armyworm, y finalmente en la clase **Healthy** tiene 3 clasificados como Armyworm.

Los TP de la clase **Blight** fueron 43 clasificados como Blight. Donde para la clase **Armyworm** obtuvo 26 clasificados como Blight, en la clase **common_rust** hubo 11 clasificados como Blight, y finalmente en la clase **Healthy** tiene 0 clasificados como Blight.

Los TP de la clase **Common Rust** fueron 44 clasificados como Common Rust. Donde para la clase **Armyworm** obtuvo 32 clasificados como Common Rust, en la clase **Blight** hubo 4 clasificados como Common Rust, y finalmente en la clase **Healthy** tiene 0 clasificados como Common Rust.

Los TP de la clase **Healthy** fueron 34 clasificados como Healthy. Donde para la clase **Armyworm** obtuvo 46 clasificados como Healthy, en la clase **Blight** hubo 0 clasificados como Healthy, y finalmente en la clase **Common Rust** tiene 0 clasificados como Healthy.

Para los TN se procede a sumar todos los valores que no pertenecen a la fila y columna del TP.

TNArmyworm = 43 + 11 + 0 + 4 + 44 + 0 + 0 + 0 + 34

TNBlight = 71 + 0 + 3 + 32 + 44 + 0 + 46 + 0 + 34

TNCommon_Rust = 71 + 6 + 3 + 26 + 43 + 0 + 46 + 0 + 34

TNHealthy = 71 + 6 + 0 + 26 + 43 + 11 + 32 + 4 + 44

Para los FP se procede a sumar todos los valores de la columna **excepto el valor de la celda** a la que pertenece la clase.

FPArmyworm = 26 + 32 + 46

FPBlight = 6 + 4 + 0

FPCommon_Rust = 0 + 11 + 0

FPHealthy = 3 + 0 + 0

Para lo FN se procede a sumar todos los valores de la fila. Inverso al caso anterior excepto el valor de la celda a la que pertenece la clase.

FN Armyworm = 6 + 0 + 3

FN Blight = 26 + 11 + 0

FN Common_Rust = 32 + 4 + 0

FN Healthy = 46 + 0 + 0

| Matriz de Observación EfficientNetV2B0 | | | | |
|---|---|---|---|---|
| **CLASES** | MEDIDAS | | | |
| | TP | TN | FP | FN |
| **Armyworm** | 71 | 136 | 104 | 9 |
| **Blight** | 43 | 230 | 10 | 37 |
| **Common Rust** | 44 | 229 | 11 | 36 |
| **Healthy** | 34 | 237 | 3 | 46 |

*Tabla 16. Matriz de Confusión del Modelo EfficientNetV2B0*

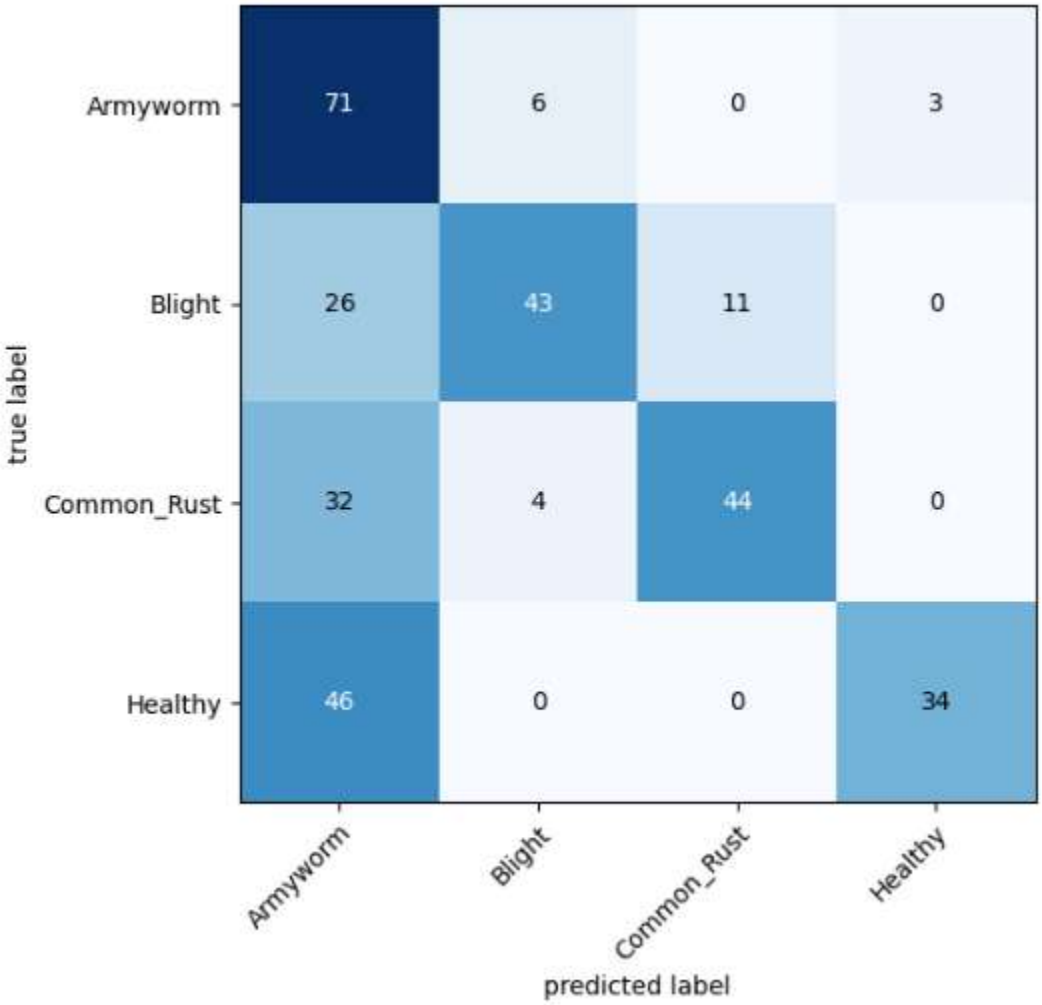



Figura 80. *Matriz de Confusión del Modelo EfficientNetV2B0*

### 5.3.4. Resultados del modelo ResNet50 en la matriz de confusión.

Los TP de la clase **Armyworm** fueron 73 clasificados como gusano. Donde para la clase **Blight** obtuvo 3 clasificados como Armyworm, en la clase **common_rust** hubo 0 clasificados como Armyworm, y finalmente en la clase **Healthy** tiene 4 clasificados como Armyworm.

Los TP de la clase **Blight** fueron 45 clasificados como Blight. Donde para la clase **Armyworm** obtuvo 33 clasificados como Blight, en la clase **common_rust** hubo 2 clasificados como Blight, y finalmente en la clase **Healthy** tiene 0 clasificados como Blight.

Los TP de la clase **Common Rust** fueron 30 clasificados como Common Rust. Donde para la clase **Armyworm** obtuvo 44 clasificados como Common Rust, en la clase **Blight** hubo 0 clasificados como Common Rust, y finalmente en la clase **Healthy** tiene 6 clasificados como Common Rust.

Los TP de la clase **Healthy** fueron 30 clasificados como Healthy. Donde para la clase **Armyworm** obtuvo 47 clasificados como Healthy, en la clase **Blight** hubo 1 clasificados como Healthy, y finalmente en la clase **Common Rust** tiene 2 clasificados como Healthy.

Para los TN se procede a sumar todos los valores que no pertenecen a la fila y columna del TP.

TNArmyworm = 45 + 2 + 0 + 0 + 30 + 6 + 1 + 2 + 30

TNBlight = 73 + 0 + 4 + 44 + 30 + 6 + 47 + 2 + 37

TNCommon_Rust = 73 + 3 + 4 + 33 + 45 + 0 + 47 + 1 + 30

TNHealthy = 73 + 3 + 0 + 33 + 45 + 2 + 44 + 0 + 30

Para los FP se procede a sumar todos los valores de la columna **excepto el valor de la celda** a la que pertenece la clase.

FPArmyworm = 33 + 44 + 47

FPBlight = 3 + 0 + 1

FPCommon_Rust = 0 + 2 + 2

FPHealthy = 4 + 0 + 6

Para lo FN se procede a sumar todos los valores de la fila. Inverso al caso anterior excepto el valor de la celda a la que pertenece la clase.

FN Armyworm = 3 + 0 + 4

FN Blight = 33 + 2 + 0

FN Common_Rust = 44 + 0 + 6

FN Healthy = 47 + 1 + 2

| Matriz de Observación ResNet50 | | | | |
|---|---|---|---|---|
| **CLASES** | MEDIDAS | | | |
| | TP | TN | FP | FN |
| **Armyworm** | 73 | 116 | 124 | 7 |
| **Blight** | 45 | 243 | 4 | 35 |
| **Common Rust** | 30 | 236 | 4 | 50 |
| **Healthy** | 30 | 230 | 10 | 50 |

*Tabla 17. Matriz de Confusión del Modelo RestNet50*

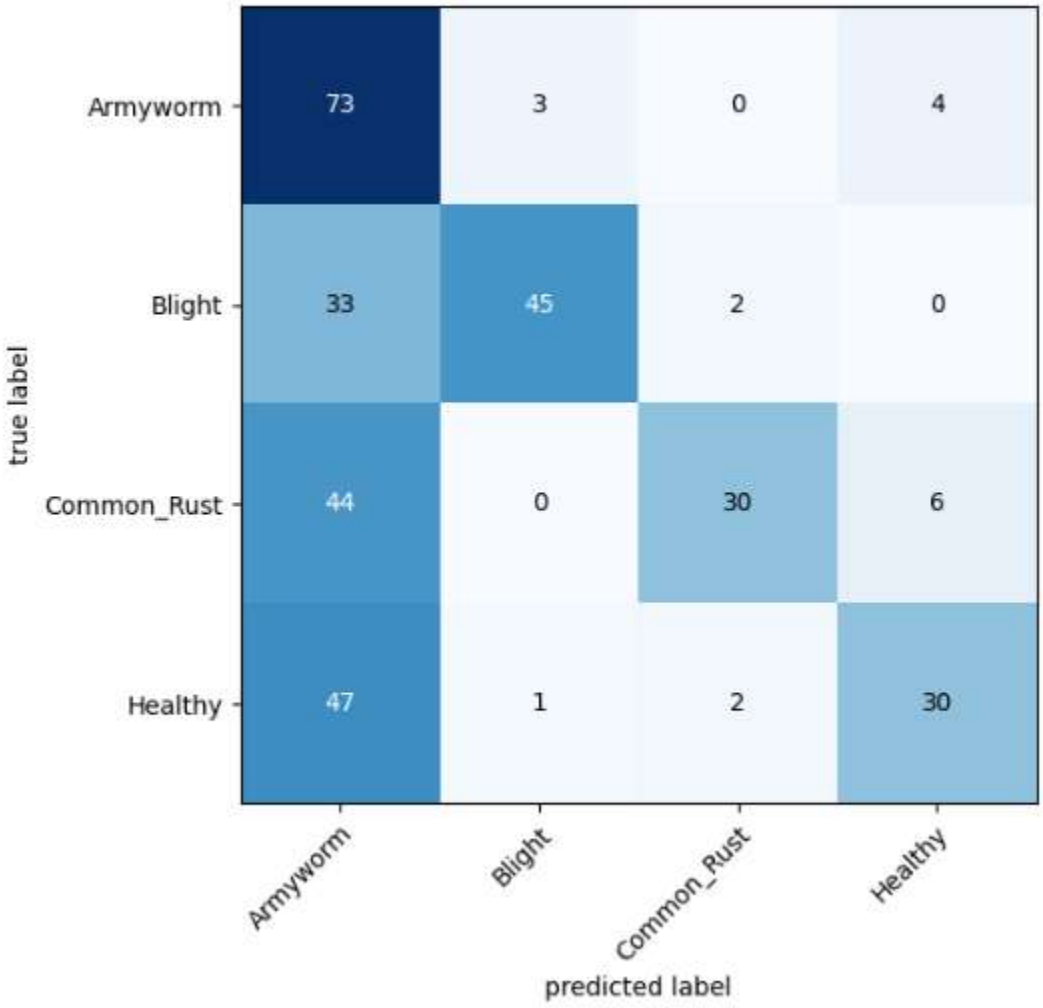



Figura 81. *Matriz de Confusión del Modelo ResNet50*

# VI. CONCLUSIONES

En el marco de los objetivos planteados, se realizó una exhaustiva Investigación del Estado del Arte de Machine Learning encontrando grandes avances tecnológicos en diferentes ámbitos como la Agricultura, la Medicina, Sistemas Climatológicos, Seguridad en sistemas de reconocimiento facial, en la Música, etc. Se tuvo que indagar diferentes terminologías, herramientas, funciones, procesos, arquitecturas, estudios en el aspecto tecnológico y también en parte de la agricultura para conocer sobre algunas enfermedades y plagas, y el proceso que realizan los agricultores para combatirlos.

En función de cada objetivo Específico se concluye:

1. Recopilar Imágenes de Plagas y Enfermedades de los cultivos de maíz:

   Se recolectó imágenes de enfermedades y plagas tanto de campo como también de repositorio kaggle, imágenes clasificadas según la enfermedad o plaga. Como son tizón de la hoja de maíz, roya común, gusano y de hojas sanas. Luego se procedió a clasificar en directorios en Google Drive.

2. Entrenar los modelos de redes neuronales utilizando técnicas de procesamiento de imágenes para la detección de plagas y enfermedades en los cultivos de maíz:

   Se desarrolló y entreno 4 modelos de redes Neuronales Convolucionales como DenseNet201, EfficientNetV2B0, MobileNetV2 y ResNet50. Modelos pre entrenados con el repositorio de imágenes ImageNet con más 10 millones y 1000 categorías de imágenes que fueron utilizadas para un preentrenamiento de modelos neuronales. Se aplicaron capas de procesamiento como aumentación, rotación, zoom y funciones y filtros que permitieron mejorar la generalización o aprendizaje de los modelos.

3. Analizar datos relevantes sobre plagas y enfermedades comunes en los cultivos de maíz en el Distrito de Cascas. Utilizando técnicas de Procesamiento de imágenes:

En función del tercer objetivo al aplicar técnicas o capas de procesamiento como aumentación de datos, funciones matemáticas en el aspecto neuronal y capas como Dropout; logramos obtener solo información relevante de una imagen, excluyendo la menos importante.

4. Evaluar los modelos para la detección de plagas y enfermedades en los cultivos de maíz. Para determinar su precisión y rendimiento en la detección en plagas y enfermedades.

   Se utilizó la matriz de confusión como herramienta de medición de desempeño de los modelos. Permitiendo identificar que tan acertados son los resultados obtenidos por cada modelo.

5. Visualizar los resultados de la predicción de cada modelo.
   Para mostrar el resultado de la precisión de los modelos se usaron librerías Python como Plotly y Matplotlib. Que permitieron tener una mejor vista del resultado de aprendizaje y precisión de cada modelo.

6. Interpretar e Identificar el mejor modelo para la detección de plagas y enfermedades en los cultivos de maíz.

   Luego de haber graficado los resultados, se procedió a pasar una imagen nueva como data de prueba o test. Para medir que tan precisos son las predicciones de cada modelo.

   A nivel de precisión el modelo DenseNet201 es más rápido para detectar, porque solo le tomo 10 epochs con una precisión de 0.988361. Es decir, un 98 %; seguido por el modelo ResNet50 con 13 epochs y una precisión de 0.990301 al 99 % en comparación a los de últimos modelos que les tomo 42 epochs.

**Objetivo General**

Identificar el modelo más eficaz de procesamiento de imágenes con Redes Neuronales para la detección de plagas y enfermedades en los cultivos de maíz en el Distrito de Cascas, La Libertad, durante el año 2022.

Con las pruebas y aplicando la técnica de medición de rendimiento de un modelo neuronal, como es la matriz de confusión se puede concluir que el modelo MobileNetV2 es mucho más preciso y cuenta con mayor cantidad de aciertos en verdaderos positivos y negativos. Sumado a ello que es un modelo bien liviano. En comparación a los demás modelos, con 3.5 millones de parámetros. Es decir, es más eficaz.

| | size_MB | parameters_M |
|---|---|---|
| ResNet50 | 50.0 | 25.6 |
| EfficienteNetV2BO | 29.0 | 7.2 |
| MobileNetV2 | 14.0 | 3.5 |
| DenseNet201 | 80.0 | 20.2 |

Figura 82. *Costo de entrenamiento MobileNetV2*

Modelo con el mejor rendimiento **MobileNetV2.**

| | | |
|---|---|---|
| DenseNet201-20230610-043821 | 0.988361 | 10 |
| EfficienteNetV2BO-20230610-061923 | 0.994180 | 42 |
| MobileNetV220230611-031218 | 0.996120 | 42 |
| ResNet50-20230610-074818 | 0.990301 | 13 |

Figura 83. *MobileNetV2*

# VII. RECOMENDACIONES

Las imágenes obtenidas no deben tomarse de un solo ángulo porque esto podría afectar al evaluar el rendimiento de los algoritmos. A si mismo si se optara por adquirir imágenes de algún dataset de internet. Estas deben tener una nitidez adecuado y un mismo tamaño en todas las imágenes.

Es recomendable tener una cantidad considerable de imágenes de cada clase que se quiera entrenar un modelo.

Para el procesamiento de imágenes se recomienda aplicar determinas capas de preprocesamiento y aumentación que son propias de la librería Keras para nuestro caso. Con la finalidad de mejorar la generalización(aprendizaje) de los modelos.

Es muy recomendable la conversión de las imágenes sea a escala de grises. Permitir la conversión de valores entre 0's y 1's para lograr un mejor entendimiento por parte del algoritmo y máquina.

Se recomienda que los modelos no sean entrenados varias veces porque esto puede ocasionar el sobre ajuste. Es decir, el modelo aprendió demasiado acerca del objetivo y esto no permite generalizar bien en otras imágenes que se le proporcione. También puede ocurrir lo contrario que el modelo tenga desajuste, quiere decir que el número de capas, funciones y filtros son muy simples o la cantidad de imágenes es muy poca. No le va a permitir aprender bien y puede ocasionar este último escenario.

Por ello se recomienda investigar que técnicas, filtros o funciones son las más adecuadas para mejorar la generalización en los modelos entrenados. También se aconseja definir determinados parámetros con un valor mínimo de acuerdo con lo que se quiera lograr.

Es recomendable contar con recursos pagados por el tema limitaciones, cuando se utiliza recursos gratuitos. Por ejemplo, para el caso de

procesamiento de imágenes se necesita GPU mínimo de 15 GB. De acuerdo con la estructura del modelo que estés desarrollando. Es decir, contar con suficientes recursos computacionales para su ejecución.

Para finalizar se recomienda aplicar una herramienta (Matriz de Confusión) que nos permita medir que tan preciso y eficaz es un modelo de red neuronal.

## VIII. REFERENCIAS BIBLIOGRÁFICAS


A Baskar, M. R. (2023). *Digital Image Processing.* Chennai, India: A Chapman & Hall Book.

Aditya Devarakonda, M. N. (2018). *ADABATCH: ADAPTIVE BATCH SIZES FOR TRAINING DEEP NEURAL NETWORKS.* Santa Clara: arXiv.

Agarwal, M., Kumar Bohat, V., Dilshad Ansari, M., Sinha, A., Kr. Gupta, S., & Garg, D. (2019). *A Convolution Neural Network based approach to detect the disease in Corn Crop.* Tiruchirappalli, India: International Conference on Advanced Computing (IACC).

Agricultura, M. d. (2016). *PLAN ESTRATÉGICO SECTORIAL MULTIANUAL.*

AI, D. (30 de Abril de 2018). *datarobot*. Obtenido de datarobot: https://www.datarobot.com/blog/introduction-to-loss-functions/

Analyticssteps. (01 de AGOSTO de 2021). *Analyticssteps*. Obtenido de Analyticssteps: https://www.analyticssteps.com/blogs/how-does-basic-convolution-work-image-processing

Aroni, H. Q. (2021). *Plagas y enfermedades en el cultivo de maíz.* PERÚ.

Awati, R. (2022). *TechTarget*. Obtenido de TechTarget: https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network

B. Chitradevi, P. (2014). *International Journal of Innovative Research in Computer and Communication Engineering.* India: ISSN.

B. Chitradevi, P. (2014). *International Journal of Innovative Research in Computer and Communication Engineering.* India: ISSN. Obtenido de https://www.rroij.com/open-access/an-overview-on-image-processing-techniques.pdf

B. Chitradevi, P. (2014). *International Journal of Innovative Research in Computer and Communication Engineering.* India: ISSN. Obtenido de https://www.rroij.com/open-access/an-overview-on-image-processing-techniques.pdf

baeldung. (6 de November de 2022). *Artificial Intelligence*. Obtenido de Artificial Intelligence: https://www.baeldung.com/cs/training-validation-loss-deep-learning#:~:text=4.,the%20performance%20of%20the%20model.

Bazán, W. C. (2016). *SERVICIO DE CONSULTORÍA PARA EL ANÁLISIS SOBRE ORGANISMOS DEL AIRE Y SUELO DEL MAÍZ.*

Beomyoung Kim, Y. Y. (2021). *Beyond Semantic to Instance Segmentation: Weakly-Supervised Instance.*

Brownlee, J. (12 de August de 2019). *Machine Learning Mastery* . Obtenido de Machine Learning Mastery : https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

Camila Laranjeira, V. F. (2021). *Machine Learning Bias in Computer Vision.* Minas Gerais. Obtenido de http://sibgrapi.sid.inpe.br/col/sid.inpe.br/sibgrapi/2021/09.06.15.19/doc/SIBGRAPI2021_Tutorial_MachineLearningBias.pdf

Chollet, F. (2021). *Deep Learning with Python.* United States of America: MANNING. Obtenido de https://cloudflare-ipfs.com/ipfs/bafykbzacebwrfndtkrq7z4cudvjnlmpxp562amuo4rsnsldvcq3cc6qxj4wc4?filename=Fran%C3%A7ois%20Chollet%20-%20Deep%20Learning%20with%20Python-Manning%20Publications%20%282021%29.pdf

Christoph H. Lampert, M. B. (2008). *Beyond Sliding Windows: Object Localization by Efficient Subwindow Search.* IEEE Xplore. doi:10.1109/CVPR.2008.4587586

Datarobot. (2022). *datarobot.* Obtenido de datarobot.: https://www.datarobot.com/wiki/fitting/#:~:text=Model%20fitting%20is%20a%20measure,doesn%27t%20match%20closely%20enough.

einfochips. (12 de Mayo de 2021). *einfochips*. Obtenido de einfochips: https://www.einfochips.com/blog/understanding-object-localization-with-deep-learning/

Ganesh, S. (24 de July de 2020). *Medium*. Obtenido de Medium: https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f

Gaudenz, N. (2021). *viso.ai*. Obtenido de viso.ai: https://viso.ai/computer-vision/image-classification/

GeeksforGeeks. (20 de Julio de 2021). *GeeksforGeeks*. Obtenido de GeeksforGeeks: https://www.geeksforgeeks.org/digital-image-processing-basics/

geeksforgeeks. (24 de 08 de 2022). *geeksforgeeks*. Obtenido de geeksforgeeks: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

Ghayoumi, M. (2022). *Deep Learning in Practice.* India: CRC Press. Obtenido de https://www.routledge.com/Deep-Learning-in-Practice/Ghayoumi/p/book/9780367458621

Harrison Kinsley, D. K. (2020). *Neural Networks from Scratch in Python.* Kinsley Enterprise Inc.

Haykin, S. (2008). *Neural Networks and Learning Machines Third Edition.* Hong Kong Montreal: Pearson.

Hill, D. S. (2007). *Pests of Crops in Warmer Climates and Their Control.* United Kingdom: Springer.

Hossain, E. (8 de April de 2021). *Medium*. Obtenido de Medium: https://medium.com/mlearning-ai/difference-between-the-batch-size-and-epoch-in-neural-network-2d2cb2a16734

Hossein Gholamalinezhad, H. K. (2009). *Pooling Methods in Deep Neural Networks, a Review.* Iran. Obtenido de https://arxiv.org/ftp/arxiv/papers/2009/2009.07485.pdf

Hub, I. C. (19 de Agosto de 2020). *IBM Cloud Learn Hub*. Obtenido de IBM Cloud Learn Hub: https://www.ibm.com/cloud/learn/supervised-learning

IBM. (2020). *IBM CLOUD*. Obtenido de IBM CLOUD: https://www.ibm.com/pe-es/cloud/deep-learning

IBM. (17 de AGOSTO de 2020). *IBM CLOUD EDUCATION*. Obtenido de IBM CLOUD EDUCATION: https://www.ibm.com/cloud/learn/neural-networks

Jiawei Han, M. K. (2012). *Data Mining.* United States of America: Elsevier.

Jinzhu Lu, L. T. (2021). Review on Convolutional Neural Network (CNN) Applied to. *agriculture*, 18.

John D. Kelleher, B. M. (2015). *FUNDAMENTALS OF MACHINE LEARNING FOR PREDICTIVE DATA ANALYTICS.* Massachusetts, EE.UU: Library of Congress Cataloging-in-Publication Data.

Josh Patterson, A. G. (2017). *Deep Learning.* United States of America: O'Reilly Media. Obtenido de https://cloudflare-ipfs.com/ipfs/bafykbzacebvlahg56eqtig43tb53h6lwthrf5mn6kfjdq43covdd6ov

7y645g?filename=Josh%20Patterson%2C%20Adam%20Gibson%20-%20Deep%20Learning_%20A%20Practitioner%E2%80%99s%20Approach-O%E2%80%99Reilly%20Media%20%282017%29.pdf

Juan R. Terven, D. M. (2023). *LOSS FUNCTIONS AND METRICS IN DEEP LEARNING.* México: Computer Science.

Karan, R. (6 de Diciembre de 2021). *naukri learning*. Obtenido de naukri learning: https://www.naukri.com/learning/articles/kdd-in-data-mining/

KEYMAKR. (08 de MAYO de 2021). *KEYMAKR*. Obtenido de KEYMAKR: https://keymakr.com/blog/instance-vs-semantic-segmentation/

Kumar, A. (2022). *Data Analytics*. Obtenido de Data Analytics: https://vitalflux.com/different-types-activation-functions-neural-networks/

Kumar, H. (2022). *Opengenus Foundation*. Obtenido de Opengenus Foundation: https://iq.opengenus.org/data-augmentation/

Kundu, R. (03 de Octubre de 2022). *v7labs*. Obtenido de v7labs: https://www.v7labs.com/blog/image-processing-guide

Lab, D. D. (2022). *Domino*. Obtenido de Domino: https://www.dominodatalab.com/data-science-dictionary/hyperparameter-tuning

Landa, J. (Febrero de 2016). *fcojlanda*. Obtenido de fcojlanda: https://fcojlanda.me/es/ciencia-de-los-datos/kdd-y-mineria-de-datos-espanol/

Maria Vakalopoulou, S. C. (2023). *Deep learning: basics and convolutional neural networks.* Paris, Francia: Hal Science. Obtenido de https://hal.science/hal-03957224/document

Mehmet Metin Ozguven, K. A. (2019). *Automatic detection and classification of leaf spot disease in sugar beet using deep learning algorithms.* Turkey: ELSEVIER.

Mehryar Mohri, A. R. (2018). *Foundations of Machine Learning.* Massachusetts: Cambridge, MA. Obtenido de https://cloudflare-ipfs.com/ipfs/bafykbzacebvaa3xypokeehslcmedmtfrulsixltcqsudgfspimbkgwfwagub6?filename=Mehryar%20Mohri_%20Afshin%20Rostamizadeh_%20Ameet%20Talwalkar%20-%20Foundations%20of%20Machine%20Learning-The%20MIT%20Press%20%282018%29.pdf

Mendoza, P. J. (2017). *El cultivo del maíz en el Mundo y en Perú.*

México, G. d. (2019). *Gobierno de México*. Obtenido de Gobierno de México: https://www.gob.mx/agricultura/articulos/la-identificacion-temprana-primera-barrera-contra-plagas-y-enfermedades-del-maiz

ncps. (2018). *NORTH COAST PHOTOGRAPHIC SERVICES*. Obtenido de NORTH COAST PHOTOGRAPHIC SERVICES: https://northcoastphoto.com/digital-explained/#:~:text=Q%3A%20What%20is%20a%20Pixel,accurate%20representations%20of%20the%20original.

Nurshazlyn Mohd Aszemi, P. D. (2019). Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms. *International Journal of Advanced Computer Science and Applications*, 10. Obtenido de https://thesai.org/Downloads/Volume10No6/Paper_38-Hyperparameter_Optimization_in_Convolutional_Neural_Network.pdf

Nyuytiymbiy, K. (30 de Diciembre de 2020). *Towards Data Science*. Obtenido de Towards Data Science: https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac

ONU. (2021). *NACIONES UNIDAS ONU*. Obtenido de NACIONES UNIDAS ONU: https://news.un.org/es/story/2021/06/1492762

opengenus. (2022). *OpenGenus Foundation*. Obtenido de OpenGenus Foundation: https://iq.opengenus.org/fully-connected-layer/

Petrou, M. P. (2010). *Image Processing: The Fundamentals, Second Edition.* Chennai, India: A John Wiley and Sons.

Prince, S. J. (2010). *Computer Vision.* CAMBRIDGE. Obtenido de https://cloudflare-ipfs.com/ipfs/bafykbzaceav3zv6l3a6v2hpyej5lgpfkfoku3l2axnhf4wboaf2qnazuskiae?filename=Dr%20Simon%20J.%20D.%20Prince%20-%20Computer%20Vision_%20Models%2C%20Learning%2C%20and%20Inference-Cambridge%20University%20Press%20%282012%29.pdf

Ramar Ahila Priyadharshini, S. A. (2019). *Maize leaf disease classification using deep convolutional neural networks.* London: Neural Computing & Applications.

Riego, M. d. (2018). *Ministerio de Desarrollo Agrarío y Riego*. Obtenido de Ministerio de Desarrollo Agrarío y Riego: https://www.midagri.gob.pe/portal/22-sector-agrario/vision-general/190-problemas-en-la-agricultura-peruana

Rikiya Yamashita, &. M. (2018). *Convolutional neural networks: an overview and application in radiology.* SpringerOpen. Obtenido de https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9

Rosebrock, A. (14 de October de 2019). *imagesearch*. Obtenido de imagesearch: https://pyimagesearch.com/2019/10/14/why-is-my-validation-loss-lower-than-my-training-loss/

rubiales, A. (16 de Octubre de 2020). *Alberto rubiales*. Obtenido de Alberto rubiales: https://rubialesalberto.medium.com/explicaci%C3%B3n-funciones-de-activaci%C3%B3n-y-pr%C3%A1ctica-con-python-5807085c6ed3

Salvador Gutiérrez, I. H.-N. (2021). *Deep learning for the differentiation of downy mildew and spider mite in grapevine under field conditions.* Spain: ELSEVIER.

Senasa. (2017). *METODOLOGÍA DE EVALUACIÓN DE PLAGAS.*

SHARMA, S. (09 de SEPTIEMBRE de 2017). *Towards Data Science*. Obtenido de Towards Data Science: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

SHARMA, S. (23 de September de 2017). *Towards Data Science*. Obtenido de Towards Data Science: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

SimpliLearn. (23 de Noviembre de 2022). *SimpliLearn*. Obtenido de SimpliLearn: https://www.simplilearn.com/image-processing-article

Sineace. (2020). *Caracterización de la Región la Libertad.*

Singh, A. (8 de May de 2020). *analyticsvidhya*. Obtenido de analyticsvidhya: https://www.analyticsvidhya.com/blog/2020/02/mathematics-behind-convolutional-neural-network/

Suresh, A. (17 de Nov de 2020). *medium*. Obtenido de medium: https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5

Suresh, A. (17 de November de 2020). *Medium*. Obtenido de Medium: https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5

Tarud, J. (23 de Noviembre de 2022). *koombea*. Obtenido de koombea: https://www.koombea.com/blog/knowledge-discovery/

techopedia. (2022). *Knowledge Discovery in Databases*. Obtenido de Knowledge Discovery in Databases: https://www.techopedia.com/definition/25827/knowledge-discovery-in-databases-kdd

Trask, A. W. (2019). *Deep Learning.* EE.UU.

Turing. (2022). *Turing*. Obtenido de Turing: https://www.turing.com/kb/necessity-of-bias-in-neural-networks

VERMA, Y. (19 de SEPTIEMBRE de 2021). *Analyticsindiamag*. Obtenido de Analyticsindiamag: https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/

Vicente N. Páliz S., J. R. (s.f.). *Plagas del Maiz.* Ecuador.

Vinayak Bharadi, M. N. (2017). Image Classification Using Deep Learning. *International Journal of Engineering Research & Technology (IJERT)*, 3. Obtenido de https://www.ijert.org/research/image-classification-using-deep-learning-IJERTV6IS110016.pdf

Woods, R. C. (2018). *Digital Image Processing.* England: Person.

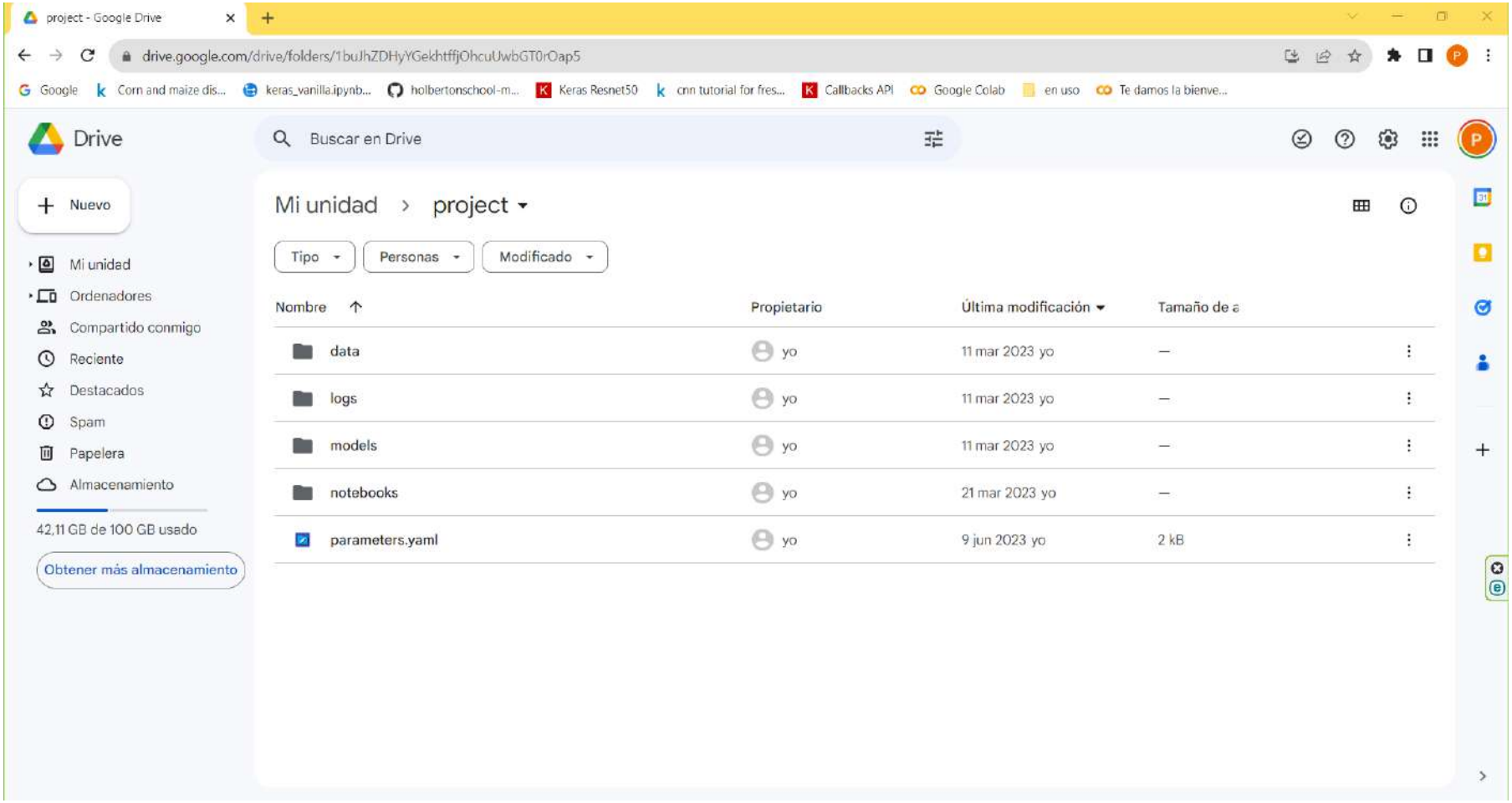
Yadav, H. (04 de Julio de 2022). *Towards Data Science*. Obtenido de Towards Data Science: https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9

Yanfen Li, H. W.-N. (2020). *Crop pest recognition in natural scenes using convolutional neural networks.* Tehran, Iran: ELSEVIER.

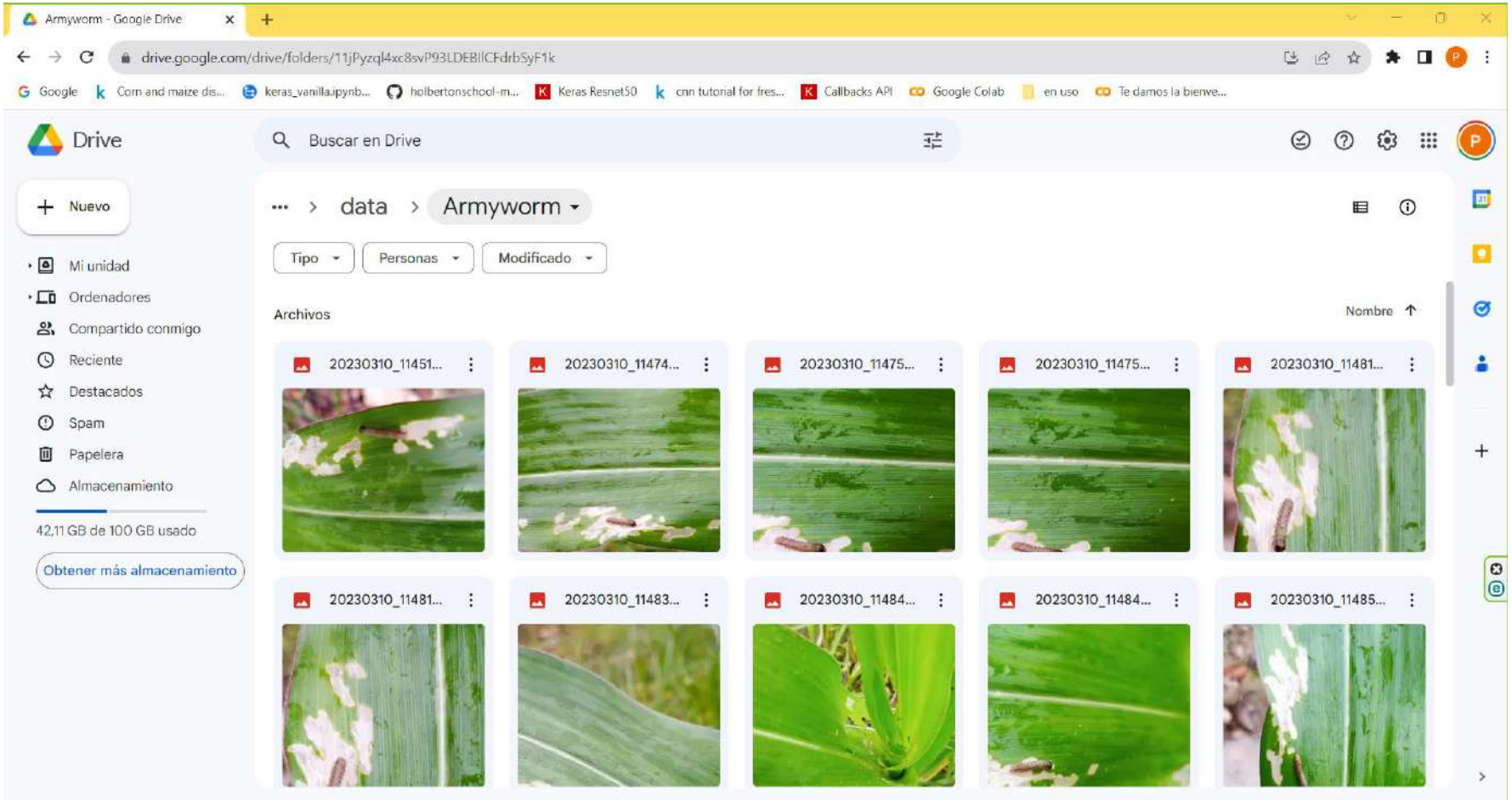
Zachte, E. (2003). *Neural Networks and Introduction to.*

## IX. ANEXOS

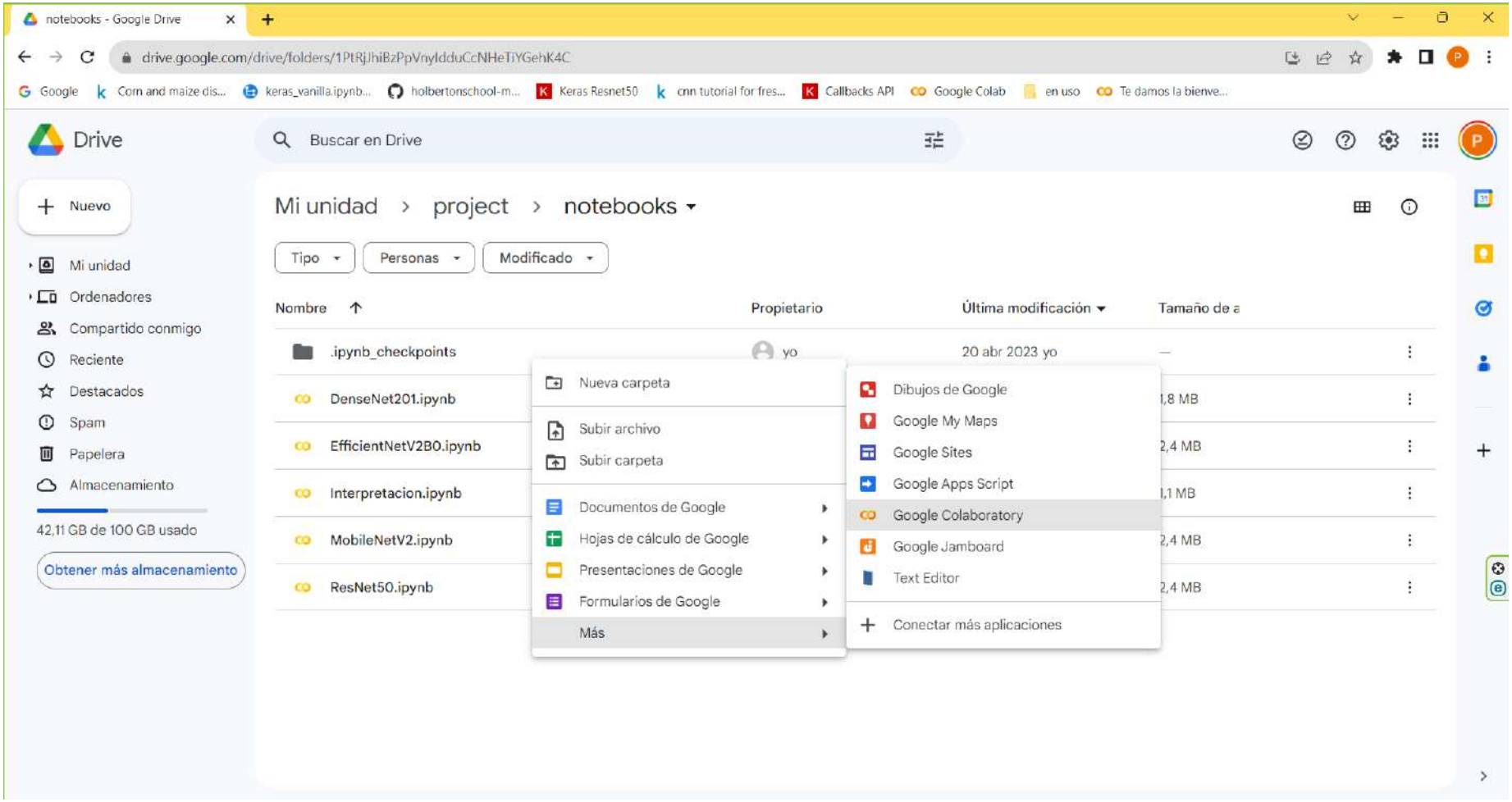
Plataforma de Google Drive donde se encuentra almacenado el Proyecto de Tesis (ANEXO 1)



Dataset de imágenes (ANEXO 2)



Creación de las notebooks en la plataforma Google Colab (ANEXO 3)



Código fuente de la notebook MobileNetV2 (ANEXO 4)