CLASIFICACIÓN AUTOMATIZADA DE ENFERMEDADES EN HOJAS DE PLANTAS USANDO REDES NEURONALES CONVOLUCIONALES (CNN)

Reporte final
21 de mayo del 2025.

Materia: Inteligencia Artificial

Alumnos:

- José Eduardo Conde Hernández (299506)
- Jair Alejandro Gaytán Espíndola (353205)
- Leonardo Trevizo Herrera (348814)

Profesor: Manuel Alberto Chávez Salcido.

Antecedentes

En los últimos años, la combinación de visión por computadora y aprendizaje profundo ha generado avances notables en la agricultura de precisión. Existen trabajos previos que usan CNNs para detectar enfermedades en cultivos como tomate, papa y maíz, con resultados alentadores. Plataformas como Kaggle han contribuido ofreciendo datasets organizados y etiquetados que permiten el entrenamiento de modelos con alto desempeño.

Justificación

Las enfermedades en plantas representan una de las principales causas de pérdida en la producción agrícola a nivel mundial. Una detección temprana y precisa es fundamental para mitigar estos daños y asegurar la calidad de los cultivos. Sin embargo, este proceso requiere experiencia técnica y puede ser subjetivo.

El presente proyecto busca automatizar este diagnóstico a través de técnicas de inteligencia artificial, específicamente redes neuronales convolucionales (CNN), que han demostrado excelentes resultados en tareas de clasificación de imágenes. Esta herramienta podría ser utilizada por agricultores o ingenieros agrónomos como un sistema de apoyo en la toma de decisiones, reduciendo costos, tiempos y errores humanos.

Objetivos

Objetivo General

Desarrollar un modelo de aprendizaje profundo capaz de clasificar automáticamente enfermedades en hojas de plantas a partir de imágenes de las mismas.

Objetivos específicos

- Implementar una arquitectura CNN eficiente para clasificación de imágenes de hojas.
- Evaluar el rendimiento del modelo usando métricas como precisión, recall y F1-score.
- Aplicar técnicas de preprocesamiento y aumentación de datos para mejorar la generalización.
- Optimizar el modelo mediante el ajuste de hiper-parámetros y técnicas como early stopping.

Hipótesis

Si se entrena una red neuronal convolucional con un dataset equilibrado y representativo de hojas enfermas, entonces el modelo podrá clasificar correctamente las enfermedades con una precisión comparable a la de un experto humano.

Metodología

Adquisición del dataset

Se utilizó un conjunto de datos público de Kaggle llamado Leaf Disease Dataset Combination, que incluye más de 15,000 imágenes clasificadas por tipo de enfermedad y con una resolución originalmente RGB 256 x 256. El conjunto de datos ya incluye splits de entrenamiento/validación/prueba.

Preprocesamiento

Antes de entrenar el modelo, se realiza una etapa de preprocesamiento de las imágenes para normalizar los datos y aumentar la diversidad del conjunto de entrenamiento, lo cual mejora la generalización del modelo. Se realiza el siguiente preprocesamiento:

- Reescalado de imágenes a tamaño uniforme (128x128 píxeles).

- Normalización de valores de píxeles al rango [0, 1].
- Aumentación de datos mediante ImageDataGenerator para cargar las imágenes en lotes durante el entrenamiento.
- Rotación aleatoria de hasta 20 grados.
- Desplazamiento horizontal y vertical.
- Se aplica ‘shear’ o se corta de manera aleatoria.
- Se aplica zoom aleatorio.
- Se invierte de manera horizontal.
- Se rellena automáticamente los bordes

Esto genera nuevas variantes de las imágenes en tiempo real durante el entrenamiento, haciendo el modelo más robusto. Todo con la finalidad de evitar, lo más posible, que el modelo ‘memorice’ los datos, lo que es importante para nosotros es obligar al modelo a aprender, que reconozca los patrones, no solo los memorice.

Entrenamiento y evaluación del modelo

Se realizaron demasiadas variantes, tanto de la definición del modelo, su compilación, preprocesamiento y su entrenamiento. Sin embargo, una de las partes más importantes es el entrenamiento, sobre todo porque es en este momento donde más recursos son consumidos durante el proceso, tanto tiempo como poder de procesamiento (los cuales son los más costosos, por ende, los más importantes). Es por eso que después de todas las variantes por las que pasamos, se concluyó en dejar únicamente un entrenamiento de 10 épocas, de tal manera, que no hubo cabida para una callback de earlñy stopping. Durante el entrenamiento se calcula la precisión y pérdida tanto en el conjunto de entrenamiento como en el de validación. Se espera que la precisión aumente y la pérdida disminuya conforme el modelo aprende, y que la validación permita detectar si el modelo está sobre ajustando.

Marco Teórico

Las Redes Neuronales Convolucionales (CNN) son un tipo de arquitectura de deep learning especialmente diseñada para procesar datos con estructura de cuadrícula, como imágenes. Su diseño se

inspira en el sistema visual humano y han demostrado un rendimiento excepcional en tareas de visión por computadora. Están compuestas por:

- Capas Convolucionales: Detectan patrones locales como bordes, colores y texturas. Utilizan filtros (kernels) que se deslizan sobre la imagen para detectar patrones locales, cada filtro aprende características diferentes (bordes, texturas, etc.). Su operación fundamental consiste en una suma ponderada de píxeles en la ventana del filtro
- Pooling: Reduce la dimensión de las imágenes intermedias conservando la información esencial. En este modelo, se usa GlobalAveragePooling2D, que calcula el promedio de cada filtro y entrega un vector resumen. Entre los beneficios existentes están: disminuir la carga computacional y proporcionar invariancia a pequeñas traslaciones.
- Funciónes de activación: MobileNetV2 utiliza funciones no lineales como ReLU6, que ayudan al modelo a representar relaciones complejas. En la capa final, se utiliza softmax, que transforma la salida en probabilidades para cada clase.
- Capas fully connected: Las capas fully conected o dense, conectan todos los valores resumidos a una predicción final. Es aquí donde el modelo “decide” a qué clase pertenece la imagen.
- Capas Dropout: Desactivan ciertas neuronas de manera aleatoria, con la finalidad de hacer el modelo dependiente de unas u otras neuronas, así evitamos el sobreajuste. En nuestro caso, estamos desactivando aleatoriamente 30% de las neuronas.
- Proceso de entrenamiento: dentro del proceso de entrenamiento, podemos establecer la función de pérdida, utilizamos Cross-entropy, utilizada para problemas de clasificación. La optimización se da mediante algoritmos como Adam ajustan los pesos para minimizar la pérdida. Dentro tenemos Backpropagation, que basicamente propaga el error hacia atrás para actualizar los parámetros

Diseño e Implementación

El proyecto fue implementado en Google Colab usando Python y TensorFlow. Basicamente el modelo de la red convolucional, tal cual, fue realizado siguiendo los patrones vistos en clase. Sin embargo, antes del diseño del algoritmo, el preprocesamiento y su entrenamiento, es importante mencionar la redacción y flujo del código. El flujo fue el siguiente:

- Descarga y preparación del dataset.
- Creación de generadores de datos para cargar imágenes por lotes.
- Definición de la arquitectura CNN.
- Entrenamiento del modelo con validación.
- Evaluación de la precisión y guardado del modelo final.

Algoritmo

Definición del modelo

Se emplea un algoritmo de transfer learning: en lugar de entrenar una red desde cero, se reutiliza un modelo preentrenado (MobileNetV2) que ya "sabe ver", y se le añade una nueva cabeza adaptada al problema actual.

- MobileNetV2 es una CNN compacta y eficiente entrenada con millones de imágenes en el dataset ImageNet.
- Se "congela" o se ignora para no volver a entrenar sus pesos, y evitar de esta manera interferir en su precisión actual.
- Se añade una cabeza personalizada, o un conjunto de neuronas para que trabajo sobre estas en la implementación actual, para poder determinar la condición de las plantas, que consite en:
    - GlobalAvaragePooling2D reduciendo la dimensionalidad.
    - Dropout, desactivando 30% de neuronas.

- Capa Dense (totalmente conectada) con una función de activación softmax y 44 salidas (número total de clases presentes en el dataset)

Compilación del modelo

Toda la carnita del modelo, o lo más interesante se encuentra la mayoría de las veces en la definición del modelo, con la caracterización de las neuronas. Sin embargo, de la compilación del modelo, se destaca que en nuestro caso se está utilizando el optimizador Adam con un grado de aprendizaje del 0.001. También, tenemos en la función de pérdida la categorical crossentropy, pues es un problema de clasificación categórica, donde las clases están codificadas en one-hot. Toma en cuenta durante el entrenamiento el porcentaje de predicciones correctas, únicamente. Además se mide la precisón en el histórico del desempeño al entenar el modelo.

Entrenamiento del modelo

El entrenamiento se configuró para ejecutarse durante 10 épocas, con una cantidad de pasos por época calculada en función del tamaño del lote y el total de muestras. De igual forma, se incluyó un conjunto de validación proporcionado por otro generador, con sus respectivos pasos calculados del mismo modo. Una vez finalizado el entrenamiento, el modelo se guardó en formato .h5 para su reutilización y posteriormente se evaluó en el conjunto de validación, reportando métricas como pérdida y precisión.

Conclusión

Durante este proyecto aprendimos muchísimo, no solo sobre cómo entrenar un modelo con redes neuronales, sino también sobre todo lo que implica llevar ese modelo a una aplicación real que otras personas puedan usar. Usamos una CNN para detectar enfermedades en hojas de plantas a partir de imágenes, y al final lo integramos todo en una web app hecha en Streamlit, donde con solo subir una imagen puedes obtener una predicción y la seguridad con la que te da la respuesta. A lo largo del proceso, entendimos mejor cómo se prepara un dataset, cómo se entrena un modelo, y lo importante que es preprocesar bien los datos. También aprendimos sobre errores comunes, cómo evitar que el modelo se sobreentrene y evitar que empiece a memorizar, y cómo presentar los resultados de forma sencilla para el

usuario. Más allá del código, nos llevamos una idea más clara de lo que implica hacer un proyecto de inteligencia artificial de principio a fin.

Implementación del modelo

En esta sección voy a aprovechar para explicar la aplicación web, pues considero que si bien el documento está enfocado en la materia “Inteligencia Artificial” y en el desarrollo de nuestra CNN, también merece mérito la realización de la aplicación y cómo está implementando el modelo.

La página web está hecha con el framework de Python “StreamLit”, éste es un framework para crear aplicaciones web, es muy sencillo de utilizar. La página se está hosteando conectando el repositorio a la página oficial de StreamLit share.streamlit.io. Una vez hecho esto, queda publicada la página web en internet, corriendo es nuestro caso app.py. Lo primero que hace es cargar el modelo, después te pide que dropees o selecciones el archivo desde el navegador de archivos, una vez cargado el archivo, hace una escalado a la imagen para que cumpla con las especificaciones de entrada del modelo. Una vez precargada la imagen y el modelo, se realiza la predicción, y en base al resultado, se hace un posprocesamiento sencillo, para presentar los datos de manera presentable, y así, seleccionando según la respuesta del modelo, el estatus en concreto de la planta de la imagen seleccionada.

Este tipo de experiencia no solo fortalece las bases técnicas, sino también nuestra capacidad de resolver problemas reales combinando visión computacional, programación en Python y despliegue web. En resumen, cerramos la materia con un proyecto completo: desde los datos hasta una aplicación usable por otros.

Más allá del código, este proyecto también refleja el cierre de un ciclo de aprendizaje donde nos familiarizamos con conceptos como redes neuronales convolucionales (CNN), funciones de activación, capas de pooling, regularización mediante Dropout, optimización con Adam, y estrategias de entrenamiento como el uso de categorical_crossentropy para clasificación multiclase. Aprendimos a trabajar con modelos preentrenados, a interpretar su arquitectura, a manejar datasets reales, y sobre todo, a convertir todo ese conocimiento en una herramienta concreta y funcional.

El desarrollo de nuestra aplicación para clasificar enfermedades en hojas de plantas nos permitió aplicar de forma práctica muchos de los conceptos fundamentales del aprendizaje automático, el preprocesamiento de datos y el despliegue de modelos en entornos accesibles como aplicaciones web. Utilizando MobileNetV2 como base, aprovechamos el poder del transfer learning para crear un modelo eficiente, ligero y preciso que no necesitó ser entrenado desde cero. El uso de aumentación de datos y la normalización durante el preprocesamiento fueron factores clave para mejorar la generalización del modelo, mientras que la integración con Streamlit facilitó la creación de una interfaz clara e intuitiva para el usuario final.

Bibliografía

Iranga, H. A. (2022). *Kaggle*. Obtenido de Leaf Disease Dataset (combination): https://www.kaggle.com/datasets/asheniranga/leaf-disease-dataset-combination