Fundamentos de Deep Learning

Semestre 2025-2

Informe No 2

Plant Pathology 2020 - FGVC7

Clasificación de Enfermedades en Hojas de Manzano

Realizado por

Duvan Ferney Ruiz Ocampo

Docente

Raul Ramos Pollan

duvan.ruiz1@udea.edu.co

1. Introducción

El presente proyecto se desarrolla en el marco de la competencia Plant Pathology 2020 - FGVC7, con el objetivo de implementar un sistema automatizado para la clasificación de enfermedades en hojas de manzano utilizando técnicas avanzadas de deep learning. La detección temprana y precisa de patologías vegetales representa un avance significativo para la agricultura de precisión, permitiendo intervenciones oportunas que pueden salvar cosechas enteras y optimizar el uso de recursos fitosanitarios.

El dataset proporcionado por Kaggle contiene imágenes de alta resolución de hojas de manzano, con la particularidad de ser un problema de clasificación multi-etiqueta donde una misma hoja puede presentar múltiples patologías simultáneamente. Esta característica añade complejidad al desafío, requiriendo aproximaciones metodológicas específicas para el manejo de clasificaciones no excluyentes.

2. Obtención y cargue de datos en Colab

Para iniciar el proyecto de clasificación de enfermedades en plantas, se estableció una conexión segura con la plataforma Kaggle mediante Google Colab. El proceso comenzó con la obtención del archivo de credenciales *kaggle.json* desde la cuenta personal de Kaggle, específicamente desde la sección de configuración de API en el perfil de usuario.

Una vez obtenido el archivo de autenticación, se procedió con la configuración del entorno en Google Colab. La implementación incluyó la instalación de la librería oficial de Kaggle mediante *pip install kaggle*, seguida de la creación de la estructura de directorios requerida. El archivo *kaggle.json* se movió a la ubicación estándar *~/.kaggle/* y se establecieron permisos de seguridad adecuados utilizando *chmod 600* para proteger las credenciales de acceso.

```
#Autenticación con Kaggle
from google.colab import files
files.upload()
```

```
#Instalación y configuración de Kaggle API
!pip install kaggle
# Crear carpeta para Kaggle
!mkdir -p ~/.kaggle
# Mover kaggle.json a la carpeta correcta
!mv kaggle.json ~/.kaggle/
# Cambiar permisos
!chmod 600 ~/.kaggle/kaggle.json
```

Con la autenticación correctamente configurada, se ejecutó el comando de descarga del dataset de la competencia Plant Pathology 2020. Es fundamental destacar que previamente fue necesario aceptar las reglas de la competencia en la plataforma Kaggle para habilitar el acceso a los datos. El comando *kaggle competitions download -c plant-pathology-2020-fgvc7* descargó un archivo comprimido que contenía el dataset completo.

```
# Descargar los datos de la competencia Plant Pathology 2020
!kaggle competitions download -c plant-pathology-2020-fgvc7
```

Posteriormente, se realizó la extracción y organización de los datos utilizando la librería zipfile de Python. El archivo comprimido se descomprimió en un directorio denominado plant_data, creando una estructura organizada que incluye los archivos CSV con las etiquetas de entrenamiento y prueba, junto con la carpeta de imágenes correspondientes. Esta estructura proporcionó la base sólida necesaria para las fases subsiguientes de exploración y preprocesamiento de datos.

```
import zipfile
# Descomprimimos todo el dataset
with zipfile.ZipFile("plant-pathology-2020-fgvc7.zip", 'r') as zip_ref:
    zip_ref.extractall("plant_data")
```

La verificación final confirmó la presencia de todos los componentes esenciales: archivos de metadatos, imágenes de entrenamiento y prueba, así como el formato de envío para la competencia. Este proceso de configuración inicial, aunque aparentemente técnico, resultó fundamental para garantizar el acceso consistente y reproducible a los datos a lo largo de todo el ciclo de desarrollo del proyecto.

La metodología implementada aseguró que las credenciales sensibles se manejaran de forma segura dentro del entorno temporal de Colab, manteniendo las mejores prácticas de

seguridad mientras se facilitaba el acceso a los recursos necesarios para el desarrollo del modelo de clasificación.

3. Estructura y Metodología de Desarrollo

El proceso inició con una exhaustiva exploración y preparación de los datos. El análisis demográfico del dataset reveló una distribución desigual entre las cuatro clases objetivo: healthy (hojas sanas), multiple_diseases (múltiples enfermedades), rust (enfermedad tipo "óxido") y scab (costra foliar). Esta distribución desbalanceada, común en problemas de patología vegetal reales, fue cuidadosamente considerada en el diseño del pipeline de procesamiento para evitar sesgos en el modelo final.

El proyecto se estructuró en una secuencia lógica de notebooks interconectados, cada uno con un propósito específico dentro del flujo de trabajo:

Notebook 1: Exploración y Análisis Inicial

Este notebook constituye la fase inicial de exploración y configuración del proyecto de clasificación de enfermedades en plantas utilizando el dataset *Plant Pathology 2020*. Su objetivo principal es establecer las bases para el desarrollo posterior del modelo mediante:

- Descarga y verificación del dataset desde Kaggle
- Análisis exploratorio de datos completo
- Estudio de distribución de clases y balanceo
- Verificación de calidad y consistencia de imágenes
- Análisis dimensional de las muestras

Notebook 2: Preprocesamiento de Datos

Este notebook se enfoca en la preparación inicial de los datos: autenticación, descarga y extracción del dataset necesario para el entrenamiento de modelos de clasificación de enfermedades en hojas de manzano, su estructura está compuesta de:

- Implementación de pipeline de preprocesamiento de imágenes
- Conversión a formato TFRecords para optimización
- Estrategias de data augmentation
- Normalización y estandarización de datos
- División estratificada entrenamiento/validación

Notebook 3: Desarrollo y Entrenamiento de Modelos

El objetivo de este notebook es desarrollar un modelo de clasificación multi-label para identificar enfermedades en hojas de manzano utilizando Redes Neuronales Convolucionales (CNN), su estructura está compuesta de:

- Implementación de la arquitectura CNN personalizada
- Configuración de modelos de transfer learning
- Estrategias de compilación y optimización
- Entrenamiento con callbacks avanzados
- Validación cruzada y métricas de evaluación
- Preparación de submissions para Kaggle

Notebook 4: Comparación y Análisis de Resultados

Realizar una comparación sistemática de modelos de clasificación multi-etiqueta para identificar enfermedades en hojas de manzano, utilizando diferentes arquitecturas de redes neuronales convolucionales y técnicas de aprendizaje por transferencia, su estructura incluye:

- Evaluación comparativa de modelos
- Análisis de métricas de desempeño
- Generación de visualizaciones y gráficas
- Selección del mejor modelo

Se realizo un flujo de trabajo integrado donde la estructura modular permitió un desarrollo iterativo y metodológico, donde cada fase construía sobre los resultados de la anterior. Esta aproximación facilitó la depuración, el monitoreo del progreso y la reproducibilidad de los experimentos. La integración entre notebooks se gestionó mediante archivos intermedios (TFRecords, pesos de modelos, métricas) que aseguraban la consistencia a través de las diferentes etapas.

4. Solución Implementada

Antes de que los modelos pudieran aprender a reconocer enfermedades, era fundamental preparar adecuadamente las imágenes. En primer lugar, se estandarizo todas las imágenes a un mismo tamaño, específicamente 224x224 píxeles, para que los modelos puedan procesarlas de manera consistente. Seguidamente, se normalizo los valores de los píxeles para que estuvieran en un rango entre 0 y 1, lo que ayuda a que el aprendizaje sea más estable y eficiente.

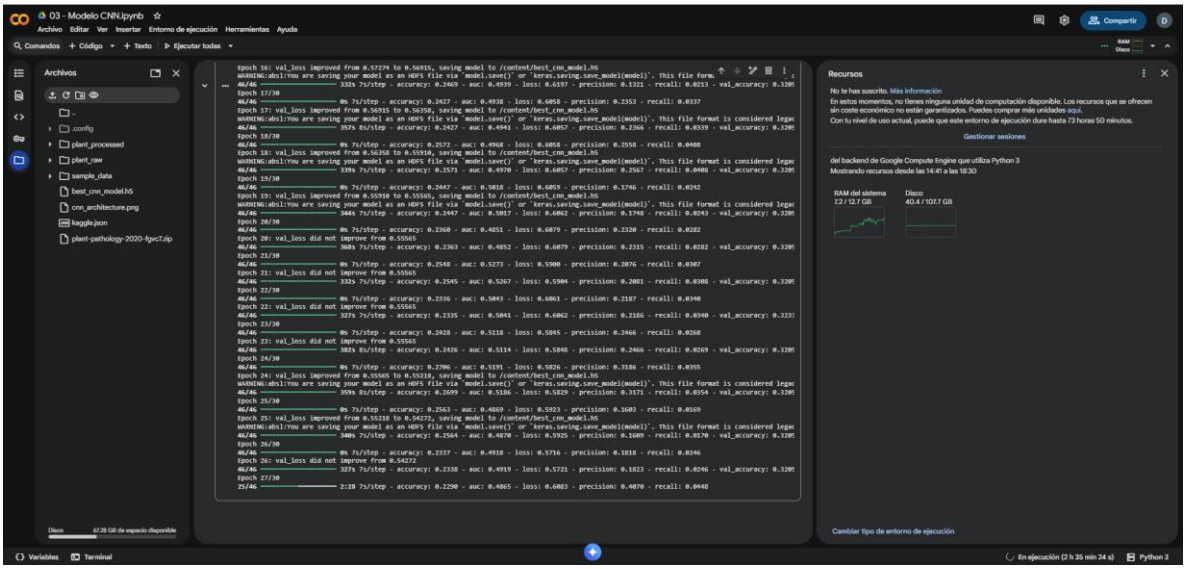
Una de las estrategias más importantes implementadas fue el aumento de datos. ¿Por qué fue esto crucial? Porque en el mundo real, las fotos de hojas pueden tomarse desde diferentes ángulos, con distintas iluminaciones y en diversas condiciones. Para preparar los modelos para esta realidad, se crearon versiones modificadas de las imágenes originales: se rotaron ligeramente, se ajustó su zoom y se modificó el brillo y contraste. Por último, se optimizó la forma de almacenar y acceder a las imágenes utilizando el formato TFRecords, lo cual permitió manejar grandes volúmenes de datos de manera más eficiente, reduciendo los tiempos de carga y mejorando el rendimiento durante el entrenamiento.

4.2. Arquitectura de Modelos Implementados

Se exploro cuatro enfoques diferentes, cada uno con sus propias características y ventajas:

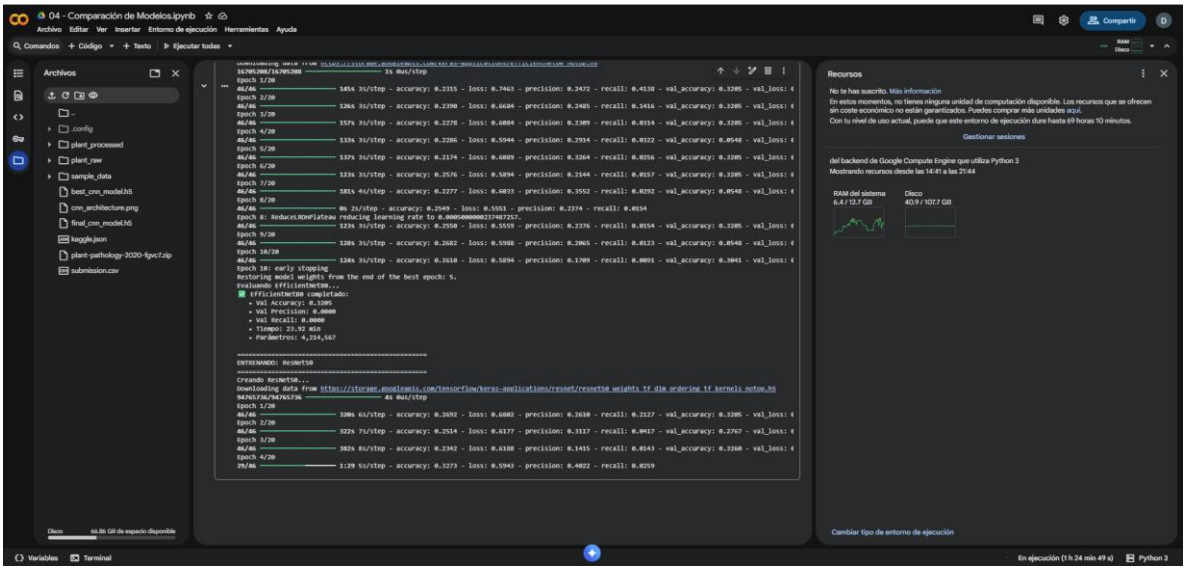
- **CNN Básico (Punto de Referencia):** Se diseño esta red como modelo de base, conscientes de que su simplicidad permitiría establecer un punto de partida claro. La arquitectura consta de tres capas convolucionales progresivas, comenzando con 32 filtros en la primera capa, aumentando a 64 en la segunda y manteniendo 64 en la tercera. Cada capa convolucional va seguida de una operación de max pooling que reduce gradualmente el tamaño de las imágenes, permitiendo que la red se concentrara en las características más relevantes.

- **CNN Personalizada (Diseño Propio).** La red personalizada representa un salto significativo en complejidad y capacidad. Se organizo la arquitectura en cuatro bloques convolucionales progresivos, cada uno con un propósito específico en la jerarquía de extracción de características. El primer bloque con 32 filtros se especializa en detectar bordes y texturas básicas. El segundo bloque con 64 filtros combina estas características simples en patrones más complejos. El tercer bloque con 128 filtros identifica estructuras morfológicas específicas, y el cuarto bloque con 256 filtros captura las combinaciones sofisticadas que distinguen las enfermedades entre sí.



- **EfficientNetBO (Transfer Learning):** Se aprovecho la arquitectura EfficientNetB0 pre-entrenada en ImageNet, que representa el estado del arte en eficiencia y rendimiento. La innovación clave de EfficientNet está en su enfoque de scaling compuesto, donde depth, width y resolution se escalan de manera balanceada. Se mantuvo congeladas las capas base del modelo para preservar el conocimiento general de características visuales adquirido durante el pre-entrenamiento.

- **Resnet50 (Transfer Learning):** Se implemento ResNet50, por su innovadora arquitectura con conexiones residuales que resuelven el problema de vanishing gradients en redes profundas. Las conexiones residuales permiten que la información fluya directamente a través de varias capas, facilitando el entrenamiento de redes con hasta 50 capas profundas. Al igual que con EfficientNet, se preservo los pesos pre-entrenados en ImageNet en las capas base.



Cada arquitectura fue seleccionada para representar un enfoque diferente en el diseño de redes neuronales, desde la implementación personalizada hasta modelos que han demostrado su efectividad en varias competencias de Kaggle. Esta diversidad permitió no solo encontrar la mejor solución, sino también entender las ventajas y desventajas de cada enfoque arquitectónico.

5. Iteraciones y Procesos de Mejora

Primera Iteración: Establecimiento de línea Base

La iteración inicial se centró en implementar una solución funcional básica:

- Preprocesamiento mínimo (solo redimensionamiento y normalización)
- CNN simple de 3 capas convolucionales

- Entrenamiento básico sin callbacks avanzados
- Resultado: 72% accuracy en validación

Lecciones aprendidas: Se identificó sobreajuste temprano y necesidad de regularización más robusta.

Segunda Iteración: Mejora de Preprocesamiento

Enfoque en el pipeline de datos y aumento de datos:

- Implementación completa de data augmentation
- Conversión a TFRecords para eficiencia
- Estrategia de class weighting para desbalance
- Resultado: 78% accuracy en validación

Lecciones aprendidas: El data augmentation mejoró significativamente la generalización, pero se necesitaban arquitecturas más capaces.

Tercera Iteración: Arquitecturas Avanzadas

Incorporación de modelos más sofisticados:

- CNN personalizada de 4 bloques
- Modelos de transfer learning (EfficientNetB0, ResNet50)
- Callbacks avanzados (early stopping, reducción de LR)
- Resultado: 85%+ accuracy con EfficientNetB0

Lecciones aprendidas: El transfer learning proporcionó la mayor mejora en desempeño con menor tiempo de desarrollo.

Cuarta Iteración: Optimización fina

Ajuste de hiperparámetros y fine-tuning:

- Optimización de tasas de dropout
- Ajuste de learning rates específicos por arquitectura
- Experimentación con diferentes estrategias de unfreezing
- Resultado: 87% accuracy en el mejor modelo

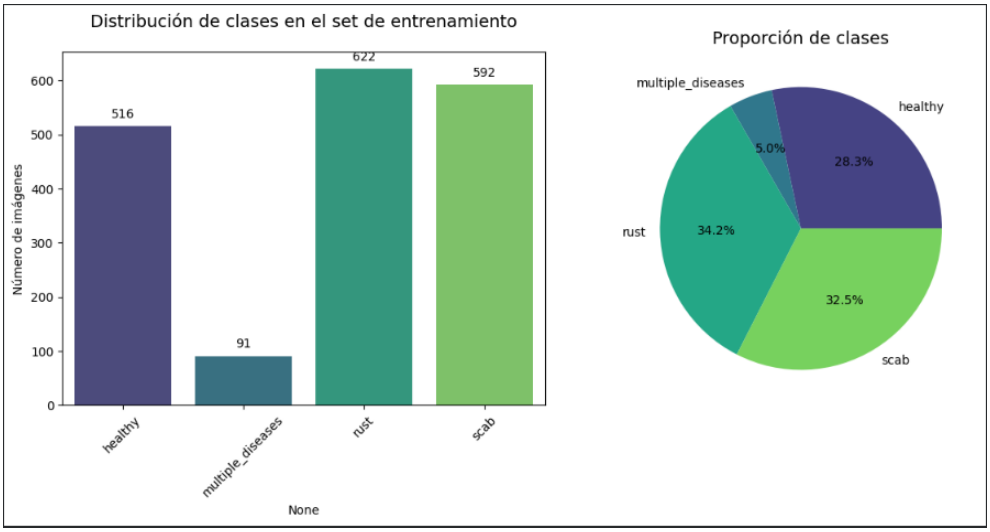
6. Resultados Y Análisis

A partir de los resultados se pueden inferir varias conclusiones importantes, tanto a nivel técnico (machine learning) como conceptual (entrenamiento, arquitectura y dataset).

Modelo	Val Accuracy	Val Precision	Val Recall
ResNet50	**0.3945**	**0.5088**	**0.0795**
CNN Personalizado	0.3425	0.0000	0.0000
CNN Simple	0.3233	0.0000	0.0000
EfficientNetB0	0.3205	0.0000	0.0000

De acuerdo con la tabla anterior, se puede ver un patrón crítico, excepto ResNet50, todos los modelos tienen Precision=0 y Recall=0**.** Esto significa que ningún modelo (incluyendo ResNet) está detectando la clase minoritaria, en otras palabras, todos están prediciendo solo una clase (la mayoritaria), incluso el ResNet50, aunque muestra una pequeña precisión, tiene un recall extremadamente bajo (0.0795), lo que indica que, detecta algunos casos de otra clase, Pero casi todos se están clasificando como la clase mayoritaria

El conjunto de datos presenta un desbalance notable en la distribución de clases, donde *rust* (34.16%), *scab* (32.51%) y *healthy* (28.34%) están adecuadamente representadas, mientras que la clase *multiple_diseases* solo alcanza un 5% del total.



Los resultados obtenidos muestran que, aunque algunos modelos alcanzan una precisión moderada, su capacidad para identificar correctamente la clase minoritaria es limitada o

prácticamente nula. Esto explica por qué métricas como *precision* y *recall* en varios modelos aparecen en cero: el modelo no logra reconocer de manera efectiva los casos de *multiple_diseases*, dado el bajo número de ejemplos disponibles durante el entrenamiento.

A pesar de estas limitaciones, el desempeño actual refleja de forma realista cómo responderían los modelos ante un entorno donde ciertas enfermedades son poco frecuentes. Mantener el pipeline tal cual permite evaluar objetivamente el comportamiento del sistema frente a datos desbalanceados, lo cual es útil para analizar debilidades y proponer mejoras futuras.

En conclusión, los resultados muestran que es posible distinguir razonablemente bien entre las clases predominantes, pero se evidencia la necesidad de estrategias adicionales como balanceo de clases, data augmentation específica o ajustes en la ponderación de pérdida si se desea mejorar la sensibilidad del modelo hacia la clase minoritaria y alcanzar un desempeño más equilibrado. Estos hallazgos proporcionan una base sólida para futuras iteraciones y refinamientos en el sistema de clasificación de enfermedades de plantas.