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

Inteligencia artificial

Integrantes

José Eduardo Conde Hernández (299506)

Jair Alejandro Gaytán Espíndola (353205)

Leonardo Trevizo Herrera (348814)

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.

Objetivo General

- 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 dentrenamiento/validación/prueba.

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.

Entrenamiento y evaluación del modelo

- Se entrenó el modelo por 10 épocas usando validación cruzada.
- Se aplicó early stopping para evitar sobreajuste.

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: Reducen la dimensionalidad espacial conservando las características más relevantes. Los tipos comunes son: max-pooling (selecciona el valor máximo) y average-pooling (promedia los valores). Entre los beneficios existentes están: disminuir la carga computacional y proporcionar invariancia a pequeñas traslaciones.
- Funciónes de activación: ReLU (Rectified Linear Unit) es la más utilizada: $f(x) = \max(0, x)$. Introduce no linealidad, esencial para representar funciones complejas, ayudando así a mitigar el problema de vanishing gradients.

- Capas fully connected: Densas y Softmax; Conectan todas las neuronas de una capa con las de la siguiente. Responsables de la clasificación final basada en las características extraídas. La última capa usa activación softmax para problemas multiclase. Basicamente toman las características extraídas y generan probabilidades de clasificación.
- 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.
- 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.

Este enfoque ha sido ampliamente validado en competiciones como ImageNet y en aplicaciones médicas, de seguridad, industria y agricultura.

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, aparte 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

Red neuronal convolucional (CNN). Primero creamos un modelo en el que vamos a ir agregando de manera secuencial capa por capa. Ahora, primero agregamos una capa convolucional de 32 neuronas con un kernel de 3x3, una función de activación ReLU, que espera una entrada de 128x128 en RGB. Luego, tenemos una capa de reducción espacial de 2x2. Posteriormente aplanamos los datos en un solo vector. Después tenemos una capa completamente conectada de 128 neuronas y una función de activación ReLU. Antes de la capa de salida, tenemos una capa de regularización, desactivando aleatoriamente 50% de las neuronas. Como capa de salida, tenemos una capa completamente conectada con 12 neuronas, una para cada clase, con una función de activación softmax.

Compilación del modelo

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. Toma en cuenta durante el entrenamiento el porcentaje de predicciones correctas, únicamente.

Entrenamiento del modelo

Antes de explicar el entrenamiento, está primero la definición del early-stopping, que vigila la pérdida en el conjunto de validación, esperando un mínimo de 10 épocas antes empezar a comparar las épocas para la precisón en validación. Una vez definida la callback de early stopping podemos proseguir con el entrenamiento, El modelo fue entrenado recibiendo como entrada un generador de imágenes preparado anteriormente con ImageDataGenerator, el cual entrega lotes de imágenes previamente reescaladas y organizadas por clase. El entrenamiento se configuró para ejecutarse durante un mínimo de 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. 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 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.

CODIGO

Clasificación de Enfermedades en Hojas de Plantas

Este notebook tiene como objetivo entrenar un modelo de clasificación de imágenes para identificar enfermedades en hojas de plantas utilizando un dataset público de Kaggle.

```
[ ]  from google.colab import files
     files.upload()
```

Mostrar el resultado oculto

```
[ ]  !mkdir -p ~/.kaggle
     !cp kaggle.json ~/.kaggle/
     !chmod 600 ~/.kaggle/kaggle.json
```

Estos comandos:

1. Aseguran que la carpeta .kaggle exista.
2. Copian el archivo de autenticación allí.
3. Le aplican permisos seguros.

Así puedes usar comandos como !kaggle datasets download ... en tu notebook para interactuar con Kaggle directamente.

Descarga y descompresión del dataset

El dataset que utilizaremos es **Leaf Disease Dataset Combination**, disponible en Kaggle. Este dataset contiene imágenes de hojas de plantas con diferentes enfermedades, organizadas en carpetas de entrenamiento, validación y prueba.

- Descargamos el dataset usando la API de Kaggle.
- Descomprimimos el archivo ZIP para acceder a las imágenes.

```
[ ] !kaggle datasets download -d asheniranga/leaf-disease-dataset-combination
```

```
Dataset URL: https://www.kaggle.com/datasets/asheniranga/leaf-disease-dataset-combination
License(s): GPL-2.0
Downloading leaf-disease-dataset-combination.zip to /content
 98% 744M/761M [00:04<00:00, 92.1MB/s]
100% 761M/761M [00:04<00:00, 195MB/s]
```

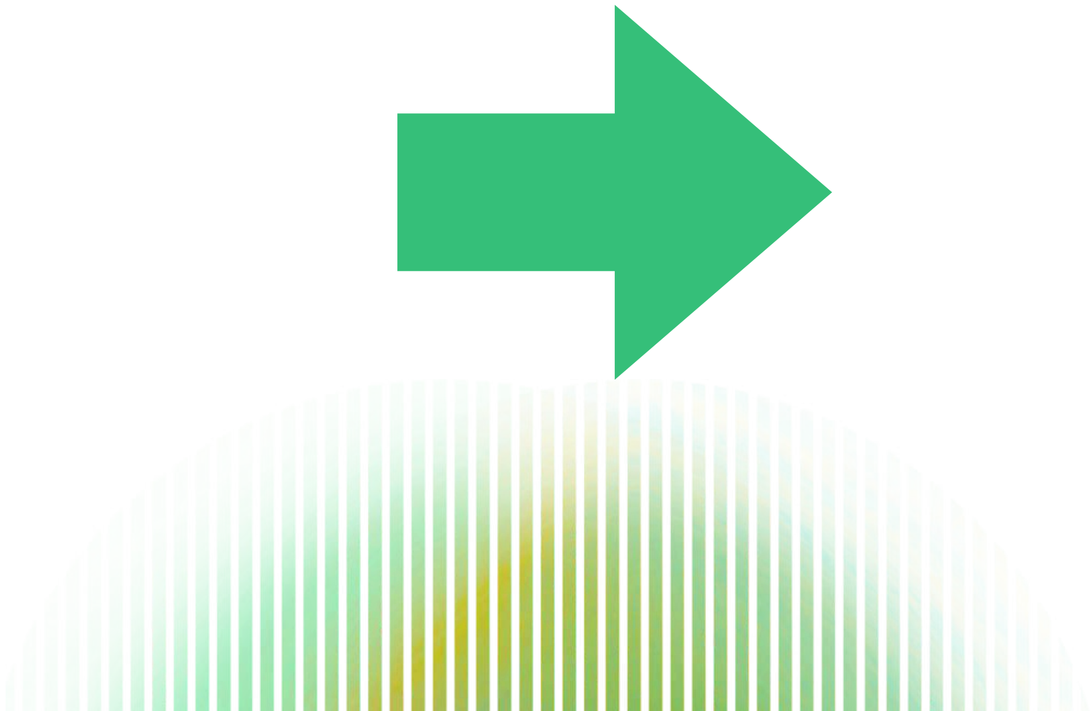
```
!unzip leaf-disease-dataset-combination.zip
```

Mostrar el resultado oculto

Preprocesamiento de imágenes

Preparamos las imágenes para el entrenamiento del modelo:

- **Normalización**: Escalamos los valores de píxeles al rango [0, 1].
- **Aumentación de datos**: Aplicamos transformaciones como rotaciones, desplazamientos y volteos para evitar overfitting.
- **Generadores de datos**: Usamos `ImageDataGenerator` para cargar las imágenes en lotes durante el entrenamiento.



Resultado

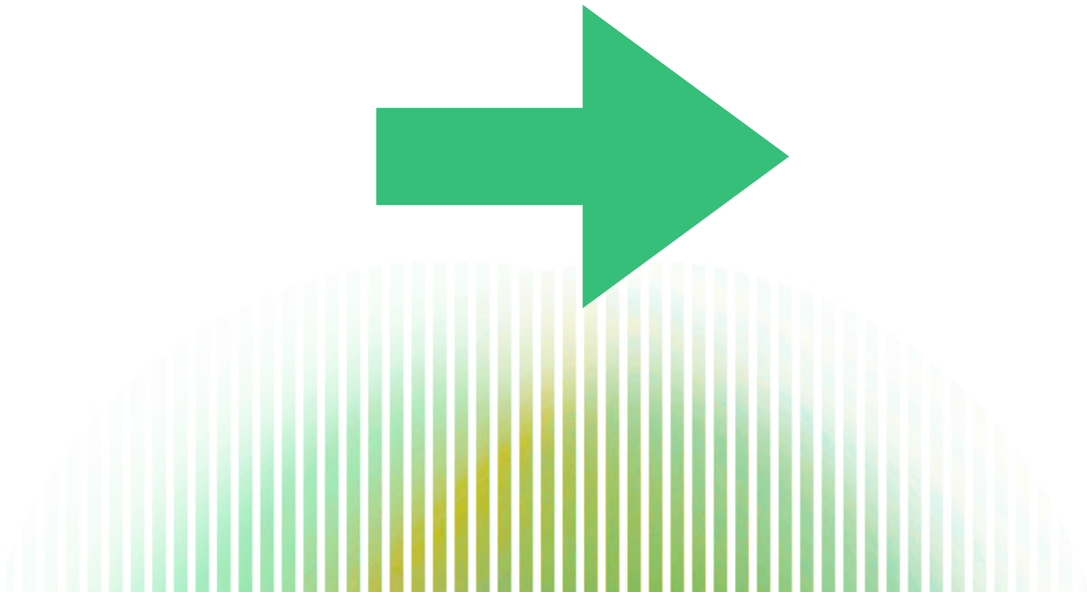
```python
import os

data_dir = '/content/image data'

print("Contenido del directorio:")
print(os.listdir(data_dir))

# Cargar imágenes
import tensorflow as tf

train_dir = os.path.join(data_dir, 'train')
validation_dir = os.path.join(data_dir, 'validation')

# ImageDataGenerator para cargar imágenes
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1.0 / 255.0)
train_generator = train_datagen.flow_from_directory(
    os.path.join(data_dir, 'train'),
    target_size=(128, 128),
    batch_size=16,
    class_mode='categorical',
    subset='training',
    seed=42
)
validation_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1.0 / 255.0)
validation_generator = validation_datagen.flow_from_directory(
    os.path.join(data_dir, 'validation'),
    target_size=(128, 128),
    batch_size=16,
    class_mode='categorical',
    seed=42
)
```

```
Contenido del directorio:
['test', 'validation', 'train']
Found 38104 images belonging to 13 classes.
Found 9458 images belonging to 13 classes.
```

Definición del modelo

Construimos una red neuronal convolucional (CNN) para clasificar las imágenes:

- **Capas convolucionales**: Extraen características de las imágenes.
- **Capas de MaxPooling**: Reducen la dimensionalidad de las características.
- **Capa densa**: Combina las características para la clasificación.
- **Capa de salida**: Usa `softmax` para predecir la probabilidad de cada clase.



```python
# Definir el modelo CNN
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(len(train_generator.class_indices), activation='softmax')
])

# Compilar el modelo
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()

# Callback para early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)
```

Resultado

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
flatten (Flatten)	(None, 127008)	0
dense (Dense)	(None, 128)	16,257,152
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 13)	1,677

```
Total params: 16,259,725 (62.03 MB)
Trainable params: 16,259,725 (62.03 MB)
Non-trainable params: 0 (0.00 B)
```

Compilación del modelo

Configuramos el modelo para el entrenamiento:

- **Optimizador**: Usamos Adam con una tasa de aprendizaje de 0.001.
- **Función de pérdida**: `categorical_crossentropy`, adecuada para clasificación multiclase.
- **Métrica**: Precisión (`accuracy`).

Entrenamiento del modelo

Entrenamos el modelo durante 50 épocas:

- **Early Stopping**: Detenemos el entrenamiento si no hay mejora en la pérdida de validación durante 10 épocas.
- **Generadores de datos**: Cargamos las imágenes en lotes para el entrenamiento y la validación.

```
[ ]  # Entrenar el modelo
     history = model.fit(
         train_generator,
         steps_per_epoch=train_generator.samples // train_generator.batch_size,
         validation_data=validation_generator,
         validation_steps=validation_generator.samples // validation_generator.batch_size,
         epochs=10,
         callbacks=[early_stopping]
     )

     # Guardar el modelo entrenado
     model.save('plant_disease_model_tf.h5')
     print("Modelo entrenado y guardado.")

     # Evaluar el modelo en el conjunto de validación
     val_loss, val_accuracy = model.evaluate(validation_generator)
     print(f"Pérdida en validación: {val_loss:.4f}")
     print(f"Precisión en validación: {val_accuracy:.4f}")
```

Resultado

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its construc
  self._warn_if_super_not_called()
2381/2381 ━━━━━━━━━━━━━━━━━━━━ 963s 404ms/step - accuracy: 0.3994 - loss: 2.3486 - val_accuracy: 0.6738 - val_loss: 0.9469
Epoch 2/10
   1/2381 ━━━━━━━━━━━━━━━━━━━━ 14:32 366ms/step - accuracy: 0.4375 - loss: 1.4365/usr/local/lib/python3.11/dist-packages/keras/src/trainers/epoch_iterator.py:107: UserWarning: Your input ran c
  self._interrupted_warning()
2381/2381 ━━━━━━━━━━━━━━━━━━━━ 39s 16ms/step - accuracy: 0.4375 - loss: 1.4365 - val_accuracy: 0.6743 - val_loss: 0.9444
Epoch 3/10
2381/2381 ━━━━━━━━━━━━━━━━━━━━ 1024s 413ms/step - accuracy: 0.5528 - loss: 1.3120 - val_accuracy: 0.6813 - val_loss: 0.8415
Epoch 4/10
2381/2381 ━━━━━━━━━━━━━━━━━━━━ 39s 16ms/step - accuracy: 0.6875 - loss: 1.3094 - val_accuracy: 0.6803 - val_loss: 0.8485
Epoch 5/10
2381/2381 ━━━━━━━━━━━━━━━━━━━━ 1041s 429ms/step - accuracy: 0.5956 - loss: 1.1370 - val_accuracy: 0.7610 - val_loss: 0.6744
Epoch 6/10
2381/2381 ━━━━━━━━━━━━━━━━━━━━ 39s 16ms/step - accuracy: 0.6250 - loss: 0.9935 - val_accuracy: 0.7611 - val_loss: 0.6791
Epoch 7/10
2381/2381 ━━━━━━━━━━━━━━━━━━━━ 1018s 410ms/step - accuracy: 0.6414 - loss: 1.0158 - val_accuracy: 0.8073 - val_loss: 0.5981
Epoch 8/10
2381/2381 ━━━━━━━━━━━━━━━━━━━━ 43s 18ms/step - accuracy: 0.8125 - loss: 0.7185 - val_accuracy: 0.8070 - val_loss: 0.5917
Epoch 9/10
2381/2381 ━━━━━━━━━━━━━━━━━━━━ 999s 410ms/step - accuracy: 0.6671 - loss: 0.9282 - val_accuracy: 0.8427 - val_loss: 0.5171
Epoch 10/10
2381/2381 ━━━━━━━━━━━━━━━━━━━━ 42s 18ms/step - accuracy: 0.6875 - loss: 0.9525 - val_accuracy: 0.8388 - val_loss: 0.5254
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Kera
Modelo entrenado y guardado.
592/592 ━━━━━━━━━━━━━━━━━━━━ 44s 74ms/step - accuracy: 0.8421 - loss: 0.5092
Pérdida en validación: 0.5170
Precisión en validación: 0.8428
```

```
from google.colab import files
files.download('plant_disease_model_tf.h5')
```

```
[ ] import tensorflow as tf

dataset = tf.keras.utils.image_dataset_from_directory(
    '/content/image data/train',
    image_size=(128, 128),
    batch_size=32,
    shuffle=False
)

class_names = dataset.class_names
print(class_names)
```

```
Found 38104 files belonging to 13 classes.
['Cassava', 'Rice', 'apple', 'cherry (including sour)', 'corn (maize)', 'grape', 'orange', 'peach', 'pepper, bell', 'potato', 'squash', 'strawberry', 'tomato']
```

Bibliografía

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