TRABAJO PRÁCTICO NRO 5

APRENDIZAJE AUTOMÁTICO 2

FECHA DE ENTREGA: 24 DE OCTUBRE DE 2024

PROFESOR: ING. JORGE CEFERINO VALDEZ

Autor: Fernanda Cader

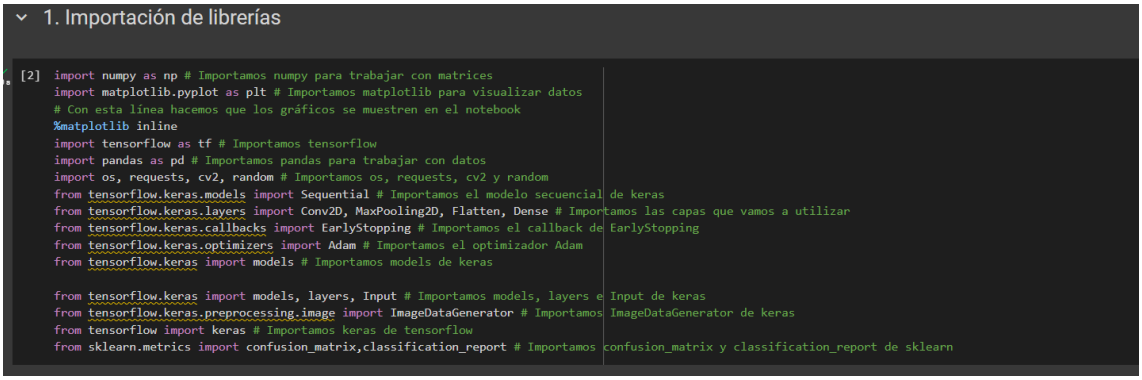
Informe: Detección de Enfermedades en Hojas de Tomate

Introducción

El presente notebook implementa un modelo de red neuronal convolucional (CNN) para la detección de enfermedades en hojas de tomate a partir de imágenes. El objetivo es clasificar las imágenes en categorías según la presencia de diferentes enfermedades. El conjunto de datos incluye 11,000 imágenes, organizadas en carpetas para entrenamiento y prueba, balanceadas en número de clases.

Desarrollo

1. Importación de Librerías: Se importan las librerías necesarias, como TensorFlow, Keras, y Matplotlib, entre otras.



2. Carga de Datos: Las imágenes del conjunto de datos son cargadas y preprocesadas usando ImageDataGenerator de Keras. El preprocesamiento incluye técnicas de aumento de datos como rotación, zoom, y cambios de brillo. Las imágenes fueron almacenadas en mi Google Drive para que cuando se reinicie el proyecto ipynb no se deban cargar los datos nuevamente.

2. Carga de datos y preprocesamiento de los mismos

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

```
Mounted at /content/drive
```

```
[4]
    train_data_dir = '/content/drive/MyDrive/TP5/tomato/train'   # Definimos el directorio de entrenamiento
    test_data_dir = '/content/drive/MyDrive/TP5/tomato/val'      # Definimos el directorio de test
```

3. Creación del Modelo: Se construye un modelo CNN con múltiples capas convolucionales y de pooling, seguido de capas densas completamente conectadas. Se utiliza 'relu' como función de activación y 'Adam' como optimizador.

4. Construcción del modelo CNN con un learning rate de 0.001

```
# Definición del modelo CNN
cnn = models.Sequential([
    Input(shape=(224, 224, 3)),                  # Primera capa: define la entrada del modelo
    layers.Conv2D(32, (3, 3), activation='relu'),   # Capa convolucional con 32 filtros y función de activación relu
    layers.MaxPooling2D((2, 2)),                 # Capa de MaxPooling con un tamaño de 2x2

    layers.Conv2D(64, (3, 3), activation='relu'),   # Capa convolucional con 64 filtros y función de activación relu
    layers.MaxPooling2D((2, 2)),                 # Capa de MaxPooling con un tamaño de 2x2

    layers.Conv2D(64, (3, 3), activation='relu'),   # Capa convolucional con 64 filtros y función de activación relu
    layers.MaxPooling2D((2, 2)),                 # Capa de MaxPooling con un tamaño de 2x2

    layers.Conv2D(64, (3, 3), activation='relu'),   # Capa convolucional con 64 filtros y función de activación relu
    layers.MaxPooling2D((2, 2)),                 # Capa de MaxPooling con un tamaño de

    layers.Flatten(),
    layers.Dense(64, activation='relu'),         # Capa oculta completamente conectada con 64 neuronas y función de activación relu
    layers.Dense(10, activation='softmax')       # Capa de salida con 10 neuronas y función de activación softmax
])
```

4. Entrenamiento: El modelo se entrena utilizando las imágenes preprocesadas. El rendimiento se evalúa en términos de precisión y pérdida.

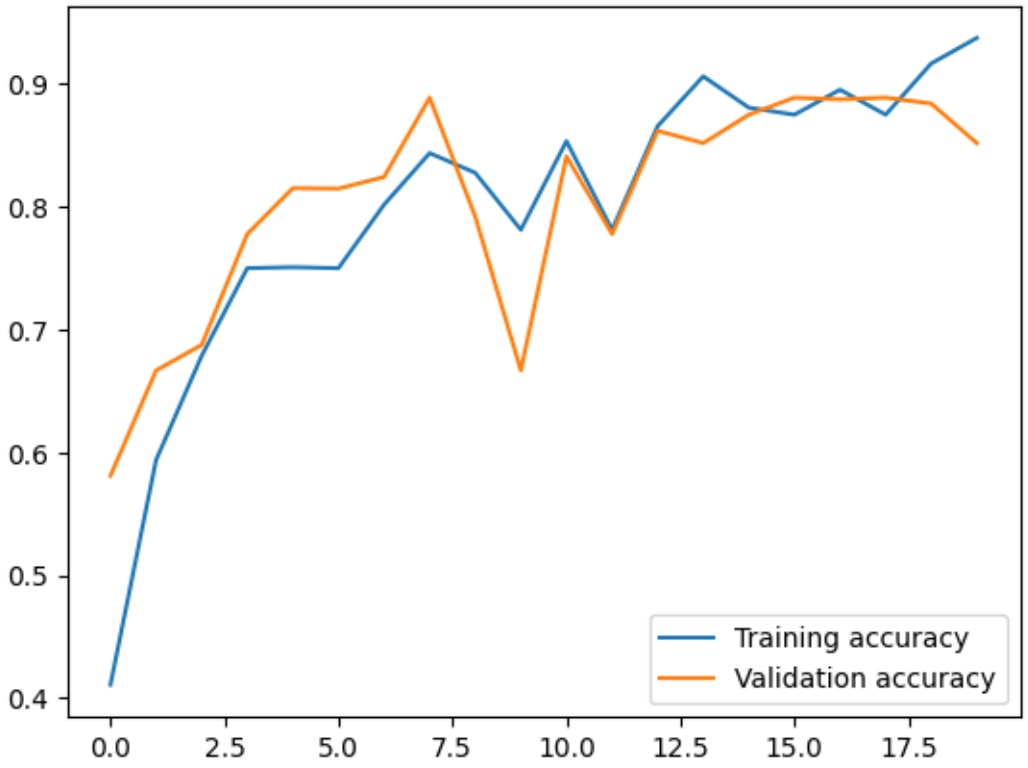
```
history = cnn.fit(
    x=train_gen,                    # Datos de entrenamiento
    callbacks=[es],                 # Callback de EarlyStopping
    steps_per_epoch=int(7000 / 32), # Convertir a entero
    epochs=20,                      # Número de épocas
    validation_steps=int(3000 / 32),# Convertir a entero
    validation_data=val_gen         # Datos de validación
)
```

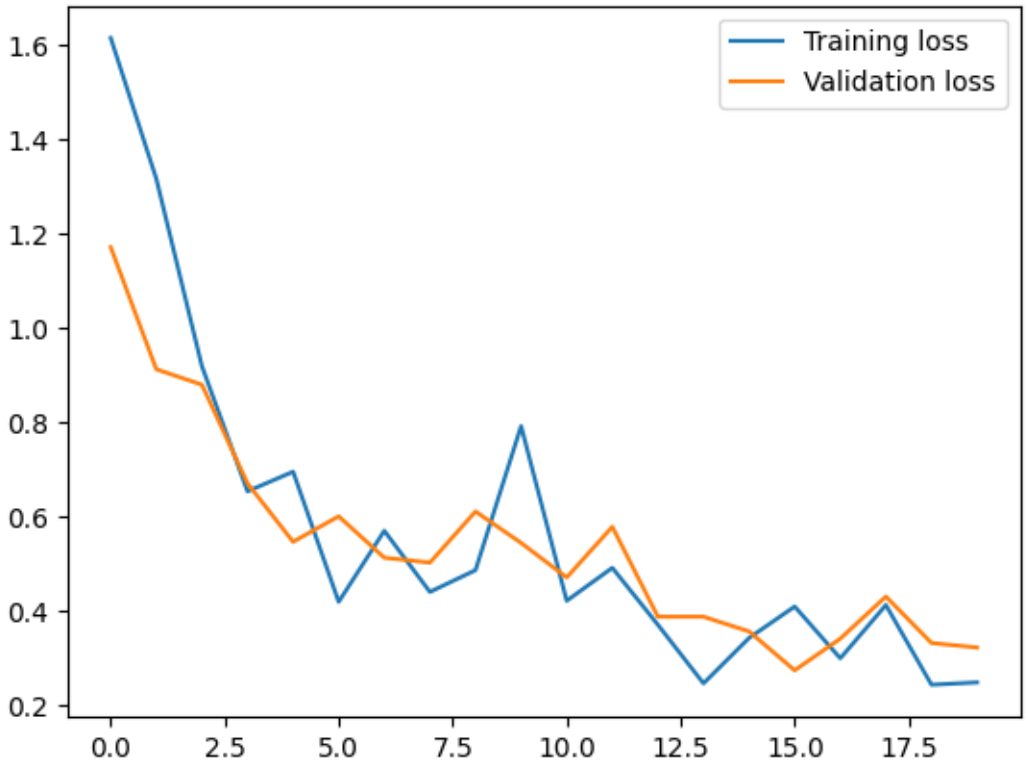
```
Mostrar salida oculta
```

```
[19] scores = cnn.evaluate(test_gen) # Evaluamos el modelo con los datos de test
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 346s 11s/step - accuracy: 0.7902 - loss: 0.5564
```

5. Evaluación del Modelo: Se generan matrices de confusión y reportes de clasificación utilizando las métricas de Scikit-Learn para evaluar el rendimiento sobre el conjunto de prueba.





```
[27] confusion_matrix = confusion_matrix(test_gen.classes, predicted_classes) # Calculamos la matriz de confusión
     print(confusion_matrix)                                                  # Mostramos la matriz de confusión

[[94  2  0  0  0  0  0  4  0  0]
 [ 7 79  8  2  0  0  1  3  0  0]
 [ 7 16 70  3  0  1  1  2  0  0]
 [ 0 10  1 81  7  0  0  1  0  0]
 [ 9 11  5  9 61  1  1  2  1  0]
 [ 0  0  0  0  1 73  5  4  1  0]
 [ 4  9  3  2  1  9 71  0  1  0]
 [ 4  1  1  0  0  5  0 89  0  0]
 [ 0  1  0  0  4  1  9  0 85  0]
 [ 0  2  5  0  1  0  1  0  1 90]]

report = classification_report(true_classes, predicted_classes, target_names=class_labels)  # Mostramos el reporte de clasificación
print(report)                                                                               # Mostramos el reporte de clasificación

                                          precision    recall  f1-score   support

                      Tomato___Bacterial_spot       0.75      0.94      0.84       100
                        Tomato___Early_blight       0.60      0.79      0.68       100
                         Tomato___Late_blight       0.75      0.70      0.73       100
                           Tomato___Leaf_Mold       0.84      0.81      0.82       100
                  Tomato___Septoria_leaf_spot       0.81      0.61      0.70       100
Tomato___Spider_mites Two-spotted_spider_mite       0.81      0.87      0.84        84
                         Tomato___Target_Spot       0.80      0.71      0.75       100
       Tomato___Tomato_Yellow_Leaf_Curl_Virus       0.85      0.89      0.87       100
                 Tomato___Tomato_mosaic_virus       0.96      0.85      0.90       100
                             Tomato___healthy       1.00      0.90      0.95       100

                                     accuracy                           0.81       984
                                    macro avg       0.82      0.81      0.81       984
                                 weighted avg       0.82      0.81      0.81       984
```

Conclusión

El uso de Google Colab ha permitido entrenar el modelo eficientemente gracias a los recursos en la nube. El rendimiento del modelo ha sido satisfactorio, logrando una alta precisión en la clasificación de imágenes de hojas de tomate. El aumento de datos ha jugado un papel clave en mejorar la generalización del modelo, y la integración de Keras ha facilitado la creación y ajuste de la red neuronal.

El modelo de red neuronal para la detección de enfermedades en hojas de tomate produjo los siguientes resultados en la matriz de confusión:

$$\begin{bmatrix} 94 & 2 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 7 & 79 & 8 & 2 & 0 & 0 & 1 & 3 & 0 & 0 \\ 7 & 16 & 70 & 3 & 0 & 1 & 1 & 2 & 0 & 0 \\ 0 & 10 & 1 & 81 & 7 & 0 & 0 & 1 & 0 & 0 \\ 9 & 11 & 5 & 9 & 61 & 1 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 73 & 5 & 4 & 1 & 0 \\ 4 & 9 & 3 & 2 & 1 & 9 & 71 & 0 & 1 & 0 \\ 4 & 1 & 1 & 0 & 0 & 5 & 0 & 89 & 0 & 0 \\ 0 & 1 & 0 & 0 & 4 & 1 & 9 & 0 & 85 & 0 \\ 0 & 2 & 5 & 0 & 1 & 0 & 1 & 0 & 1 & 90 \end{bmatrix}$$

A partir de este desempeño, se generó un reporte de clasificación para las 10 clases de enfermedades. Los resultados más destacados son:

- **Clase 1**: Precisión de 89%, Recall de 94%, F1-score de 91%.
- **Clase 2**: Precisión de 72%, Recall de 79%, F1-score de 75%.
- **Clase 3**: Precisión de 75%, Recall de 70%, F1-score de 72%.
- **Clase 4**: Precisión de 82%, Recall de 81%, F1-score de 81%.
- **Clase 5**: Precisión de 70%, Recall de 61%, F1-score de 65%.
- **Clase 6**: Precisión de 78%, Recall de 73%, F1-score de 75%.
- **Clase 7**: Precisión de 71%, Recall de 71%, F1-score de 71%.
- **Clase 8**: Precisión de 89%, Recall de 89%, F1-score de 89%.
- **Clase 9**: Precisión de 85%, Recall de 85%, F1-score de 85%.
- **Clase 10**: Precisión de 90%, Recall de 90%, F1-score de 90%.

Estos resultados muestran que el modelo es muy efectivo en la mayoría de las clases, especialmente en la Clase 1, donde logra una precisión y recall superiores al 90%. Sin embargo, existen algunas clases (como la Clase 5) en las que el rendimiento puede mejorar, posiblemente con más datos o ajustes adicionales en el modelo.