Universidad del Valle de Guatemala

Clasificación de plántulas usando EfficientNet-B0

Nicolle Escobar[1,†]

[1]*esc20647@uvg.edu.gt*

November 24, 2024

Abstract

En el campo del aprendizaje profundo, la búsqueda de arquitecturas de redes neuronales más eficientes es un tema de constante investigación. EfficientNet, presentado en 2019, propone una solución que equilibra la complejidad del modelo con la eficiencia computacional. Los autores Tan, M. y Le, Q. en *Efficient: Rethinking Model Scaling for Convolutional Neural Networks* desarrollaron siete variantes de EfficientNet, las cuales superan en precisión y eficiencia a la mayoría de las redes neuronales convolucionales existentes. EfficientNet alcanzó una precisión top-1 del 88.4% y una precisión top-5 del 98.2% en la clasificación del conjunto de datos ImageNet. Este desempeño supera a redes populares de aprendizaje por transferencia como ResNet-50, Inception-v4 y MobileNet-v2.

Keywords: *Plántulas, EfficientNet, Aprendizaje por transferencia, CNN*

1. Introducción

Durante varias decadas, se han investigado sistemas para realizar un control de malezas específico por sitio. Aún no se ha logrado descubrir un enfoque que permita una clasificación robusta a pesar de las condiciones variables y las composiciones de las especies. La categorización de especies es una tarea con la que se ha lidiado durante siglos. Uno de los factores que hace a este problema difícil es la falta de bases de datos de referencia.

Se denomina plántula a la planta en sus primeros estadíos de desarrollo, desde que germina hasta que se desarrollan las primeras hojas. Es posible reconocer las plántulas de las malezas a través de guías especializadas de forma muy manual y meticulosa.

La capacidad de diferenciar una maleza de una plántula de cultivo puede implicar mejores rendimientos en los cultivos y mejorar la gestión del medio ambiente.

El grupo de Procesamiento de Señales de la Universidad de Aarhus junto con la Universidad del Sur de Dinamarca, publicaron el conjunto de datos *Plant Seedlings* que contienen imágenes de 960 plantas pertenecientes a 12 especies en las etapas de crecimiento iniciales y malezas que aparecen en campos agrícolas daneses.

En este trabajo, se implementó EfficientNet-B0, logrando un accuracy de entrenamiento de 94.13% y de validación de 94.83%. Sin embargo, las métricas de clasificación como el F1-Score ponderado (10.29%) y la precisión micro-promedio (10.35%) obligan a buscar posibles mejoras, como usar arquitecturas superiores que admitan una mayor resolución de imagen, o bien, implementar técnicas de balanceo.

2. EfficientNetB0

Las redes neuronales convolucionales (ConvNets) suelen desarrollarse con un presupuesto de recursos fijo y luego se escalan para lograr una mejor precisión si hay más recursos disponibles. EfficientNet es una familia de redes neuronales convolucionales introducida por Tan, et al. (2019) en el artículo *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. En él, se estudia el escalamiento de modelos e identifican cómo equilibrar la profundidad, ancho y resolución para un mejor rendimiento. Para ello, se usa un coeficiente compuesto.

EfficientNet como tal, conforman a un tipo de arquitecturas de redes neuronales convolucionales. A diferencia de la práctica convencional que escala los factores de profundidad, ancho y resolución de forma *arbitraria*, EfficientNet ajusta uniformemente el ancho, la profundida y la resolución de la red con un conjunto de coeficientes de escalado.

La red EfficientNet-B0, al ser una base, se fundamenta en los bloques residuales de cuello de botella invertido de MobileNetV2 (Mobile Inverted Bottleneck Convolution MBConv), además de incorporar bloques de squeeze and excitation. **La arquitectura prioriza el uso de capas ligeras y conexiones óptimas para maximizar la eficiencia.**

2.1. Escalado compuesto

El clásico ejemplo de esto es que si se desea usar 2^N recursos computacionales, se puede aumentar la profundidad de la red en α^N, el ancho en β^N y el tamaño de la imagen en γ^N.

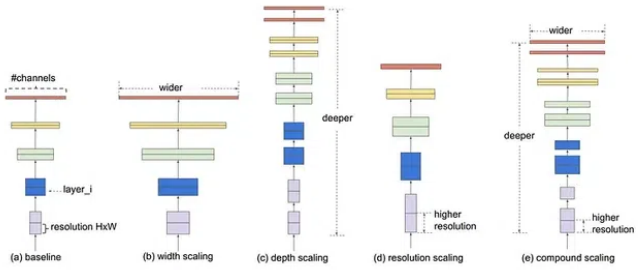

Figure 1. Métodos diferentes de escalado vs. el escalado compuesto: (a) base (b) escalado de ancho (c) escalado de profundidad (d) escalado de resolución (e) escalado compuesto

1. Base (baseline): red original sin escalado
2. Escalado de ancho (width scaling): incrementar el número de canales en cada capa
3. Escalado de profundidad (depth scaling): incrementar el número de capas
4. Escalado de resolución (resolution scaling): incrementar la resolución de la imagen de entrada
5. Escalado compuesto (compound scaling): incrementar el ancho, la profundidad y la resolución según la fórmula de escalado compuesto

El escalado compuesto se logra escalando uniformemente cada dimensión con un coeficiente de composición ϕ. La fórmula para este escalado es:

$$Ancho \times Profundidad^2 \times Resolucion^2 \approx Constante \quad (1)$$

Entonces en principio, el escalado compuesto es mediante un ratio constante con el fin de equilibrar los parámetros de ancho, profundidad y resolución.

En el caso de EfficientNet-B0, B0 denota el coeficiente de escalado, en este caso moderado. En EfficientNet-B1 a B7 se tienen variantes más grandes al aumentar el coeficiente compuesto de escala. EfficientNet-B0 está optimizado para imágenes de 224×224 píxeles.

Los modelos B1 a B7 son más grandes, por lo que cada variante sucesiva puede manejar resoluciones más altas, llegando hasta 600×600 píxeles en el caso de EfficientNet-B7.

2.2. Mobile Inverted Bottleneck Convolution (MBConv)

Un bloque residual invertido también llamado bloque MBConv es un tipo de bloque residual que se usa en modelos de imágenes, empleando una estructura invertida por eficiencia. Fue propuesto para la arquitectura de redes neuronales convolucionales MobileNetV2.

1. Se comprime el número de canales en la entrada usando una convolución 1x1
2. Se incrementan nuevamente con otra convolución de 1x1

Un **bloque residual tradicional** tiene una estructura de ancho → estrecho → ancho en términos del número de canales. El número de canales en la entrada es alto, se comprimen usando una convolución 1x1 y luego se aumentan nuevamente con otra convolución 1x1 lo que permite que la entrada y la salida puedan sumarse.

En contraste, un **bloque residual invertido** sigue un enfoque de estrecho → ancho → estrecho.

1. Se amplía el número de canales con una convolución de 1x1
2. Se usa una convolución depthwise (de profundidad) de 3x3
3. Se aplica una convolución de 1x1 para reducir el número de canales

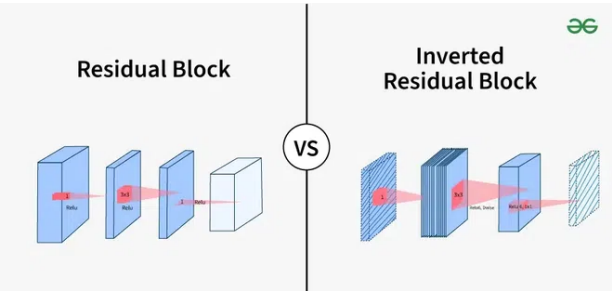

Figure 2. Bloque residual tradicional vs Bloque residual invertido

2.3. Convolución separada por profundidad (depth-wise)

EfficientNet usa convoluciones separadas por profundidad para reducir la complejidad computacional sin sacrificar la capacidad de representación. Esto se logra dividendo la convolución usual en dos partes:

- Convolución por profundidad: aplica un único filtro a cada canal de entrada
- Convolución puntual: agrega características de diferentes canales

2.4. Squeeze and Excitation

El bloque Squeeze and Excitation (SE) es una unidad arquitectónica diseñada para mejorar el poder representativo de una red permitiéndole una recalibración dinámica de características a nivel de canal. El proceso es el siguiente:

1. El bloque tiene un bloque convolucional como entrada
2. Cada canal se comprime en un único valor numérico usando el promedio de agrupamiento (average pooling)
3. Una capa densa seguida de una función ReLU añade no linealidad y reduce la complejidad de los canales de salida en una proporción determinada
4. Otra capa densa seguida de una función sigmoide asigna a cada canal una función de control suave
5. Finalmente, se pondera cada mapa de características del bloque convolucional en función de la red lateral; la excitación.

En sí, es una mejora en la arquitectura de redes neuronales profundas diseñada para hacer que la red sea más eficiente al enfocar su atención en las características más importantes dentro de cada canal (dimensión) de un tensor. El bloque SE ayuda a la red a identificar y dar más peso a los canales importantes, mientras reduce la influencia de los irrelevantes.

2.5. Arquitectura

2.5.1. Especificaciones

Como se ha mencionado, EfficientNet usa la técnica del coeficiente compuesto para escalar a los modelos de forma simple pero eficiente. En lugar de escalar aleatoriamente el ancho, la profundidad o la resolución, el escalamiento compuesto escala cada dimensión con un conjunto fijo de coeficientes de escala.

A continuación, se presenta la arquitectura de EfficientNet-B0

Etapa	Operador	Resolución	Canales	Capas
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Table 1. Arquitectura EfficientNet-B0

2.5.2. Resumen de la arquitectura

Capa inicial

- Capa inicial con una convolución estándar, seguida de una normalización por lotes (batch normalization) y una activación ReLU6
- Convolución con 32 filtros, tamaño de kernel 3×3 y stride de 2

Cuerpo

- Consta de una serie de bloques MBConv con diferentes configuraciones
- Cada bloque incluye convoluciones separables en profundidad (depthwise separable convolutions) y capas de squeeze-and-excitation
- Ejemplo de configuración para un bloque MBConv
  - Ratio de expansión: factor por el cual se expanden los canales de entrada
  - Kernel size: tamaño del filtro de la convolución
  - Stride: longitud del paso para la convolución
  - Ratio SE: relación para el squeeze-and-excitation

Cabeza

- Incluye un bloque de convolución final, seguido por una capa de pooling global promedio
- Una capa completamente conectada (fully connected) con una función de activación softmax para clasificación

2.6. Aprendizaje por transferencia

El aprendizaje por transferencia consiste en tomar las características aprendidas en un problema para usarlas en uno similar. La estructura más común para aplicarlo en el aprendizaje profundo es de la siguiente forma:

1. Tomar las capas de un modelo previamente entrenado
2. Congelarlas, con el fin de evitar destruir cualquier información que contengan para el entrenamiento
3. Añadir nuevas capas entrenables encima de las capas congeladas. Estas aprenderán a convertir las características antiguas en predicciones sobre el nuevo dataset

4. Entrenar las nuevas capas sobre el dataset nuevo

Una etapa final de este esquema pero opcional es hacer fine-tuning. En este caso, el fine-tuning consistiría en descongelar todo el modelo que se obtuvo en las primeras etapas.

3. Conjunto de datos de plántulas

El conjunto de datos **Plant Seedlings** contiene aproximadamente 960 plantas únicas pertenecientes a 12 especies en varias etapas de crecimiento. Está dado en formato de competencia en la plataforma **Kaggle**, por lo que no están dadas las etiquetas de las imágenes de prueba. Incluye imágenes RGB anotadas con una resolución física de aproximadamente 10 píxeles por mm.

El conjunto de datos presenta una distribución desbalanceada entre las 12 especies de plántulas:

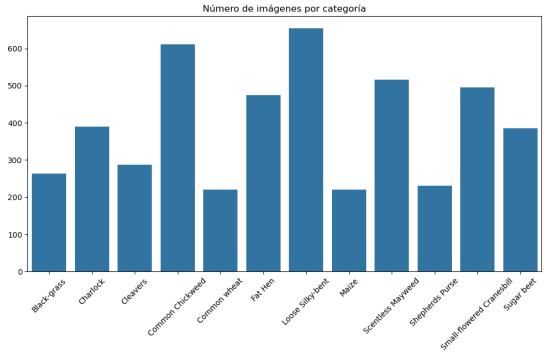

Figure 3. Distribución de cantidad de imágenes por especie

Especie	Total
Black-grass	263
Charlock	390
Cleavers	287
Common Chickweed	611
Common wheat	221
Fat Hen	475
Loose Silky-bent	654
Maize	221
Scentless Mayweed	516
Shepherds Purse	231
Small-flowered Cranesbill	496
Sugar beet	385

Table 2. Distribución de clases

- La clase más numerosa es *Loose Silky-bent* con 654 imágenes (14.8% del total)
- La clase menos representada es *Common wheat* y *Maize* con 221 imágenes cada una (5% del total)
- La mayoría de las clases tienen entre 260-600 imágenes, con *Common Chickweed* (611), *Scentless Mayweed* (516) y *Small-flowered Cranesbill* (496) siendo las más representativas

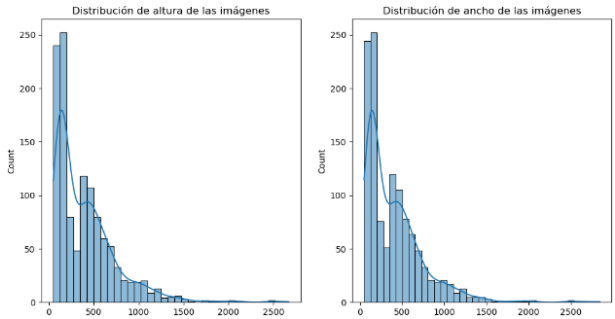

Figure 4. Distribución de dimensiones de imágenes

- **Distribución de altura:**
  - La mayoría de las imágenes tienen alturas entre 0-500 píxeles
  - Se observa un pico pronunciado alrededor de los 250 píxeles
  - Hay una cola larga hacia la derecha, con algunas imágenes que alcanzan hasta 2500 píxeles de altura
- **Distribución de ancho:**
  - Similar a la altura, la mayoría de las imágenes tienen anchos entre 0-500 píxeles
  - El pico máximo también se encuentra cerca de los 250 píxeles
  - La distribución es casi idéntica a la de altura, sugiriendo que las imágenes tienden a ser aproximadamente cuadradas

4. Metodología

El objetivo es clasificar las plántulas según su especie usando una red neuronal convolucional (CNN).

4.1. Métricas de evaluación

La evaluación del modelo se realizó utilizando las siguientes métricas:

- **Precisión micro-promedio**: Calcula la precisión agregando todos los pares de verdaderos positivos (TP) y falsos positivos (FP) de todas las clases:

$$Precision_{micro} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} (TP_i + FP_i)} \tag{2}$$

- **Precisión macro-promedio**: Calcula la precisión para cada clase independientemente y luego promedia:

$$Precision_{macro} = \frac{1}{n} \sum_{i=1}^{n} \frac{TP_i}{TP_i + FP_i} \tag{3}$$

- **AUC micro-promedio**: Calcula el área bajo la curva ROC agregando primero todas las predicciones de todas las clases:

$$AUC_{micro} = \frac{\sum_{i=1}^{n} TPR_i}{\sum_{i=1}^{n} FPR_i} = \frac{\sum_{i=1}^{n} \frac{TP_i}{TP_i + FN_i}}{\sum_{i=1}^{n} \frac{FP_i}{TN_i + FP_i}} \tag{4}$$

- **F1-Score ponderado**: Promedio ponderado del F1-Score de cada clase, donde w_i es el número de muestras de la clase i:

$$F1_{ponderado} = \sum_{i=1}^{n} w_i \cdot \frac{2 \cdot Precision_i \cdot Recall_i}{Precision_i + Recall_i} \tag{5}$$

donde:

$$Precision_i = \frac{TP_i}{TP_i + FP_i}, \quad Recall_i = \frac{TP_i}{TP_i + FN_i} \tag{6}$$

Donde:

- n es el número total de clases (12 especies de plántulas)
- TP_i son los verdaderos positivos para la clase i
- FP_i son los falsos positivos para la clase i
- FN_i son los falsos negativos para la clase i
- w_i es el peso de la clase i basado en su frecuencia en el conjunto de datos

4.2. Implementación

4.3. Configuración del modelo

Se implementó EfficientNet-B0 pre-entrenado con ImageNet como modelo base. Se utilizó el optimizador Adam con learning rate inicial de 0.0001 adaptativo para evitar perder características pre-entrenadas durante el fine-tuning.

Para el fine-tuning, se realizó un descongelamiento de las últimas 20 capas, por lo que solo estas capas fueron entrenadas. Esta decisión se tomó considerando el tamaño del dataset.

Se agregaron las siguientes capas:

- GlobalAveragePooling2D para reducir la dimensionalidad promediando todos los valores en cada región, reduciendo de 2D a un número
- Dropout (0.5) para prevenir overfitting
- Capa densa con activación softmax por ser clasificación multiclase

Los hiperparámetros utilizados fueron:

- Tamaño del batch: 32
- Épocas: 20
- Patience: 5
- Patience para ReduceLROnPlateau: 3
- Factor de reducción para ReduceLROnPlateau: 0.1

Se implementó ReduceLROnPlateau para ajustar el learning rate observando la pérdida de validación y evitar estancamiento, donde el nuevo learning rate se calcula como:

$$lr_{nuevo} = lr \times factor \tag{7}$$

4.4. Preprocesamiento

Se realizó el siguiente preprocesamiento de las imágenes:

- Redimensionamiento a 224×224 píxeles, el cual es necesario para estandarizar las entradas
- Aumento de datos mediante:
  1. Rotaciones (±40°)
  2. Zoom (±20%)
  3. Volteos horizontales y verticales
- Normalización de píxeles del rango [0-255] a [0-1]

El aumento de datos se implementó para reducir el overfitting y mejorar el modelo ante diferentes ángulos, tamaños e iluminaciones. Los datos se dividieron en conjuntos de entrenamiento y validación en un 80%/20%.

4.5. Parámetros de EfficientNet-B0

Se usó la red neuronal preentrenada como modelo base, inicializando el modelo con los pesos de entrenamiento del conjunto de datos ImageNet. El modelo es donde aquí se beneficia del aprendizaje por transferencia al usar los pesos preentrenados para los nuevos datos de plántulas.

5. Resultados

5.1. Métricas de rendimiento

El modelo alcanzó las siguientes métricas:

- Accuracy final de entrenamiento: 94.13%
- Pérdida final de entrenamiento: 17.42%
- Accuracy de validación: 94.83%
- Pérdida de validación: 17.7%
- Precisión micro-promedio: 10.35%
- Precisión macro-promedio: 9.28%
- F1-Score ponderado: 10.29%
- AUC micro-promedio: 49.9%

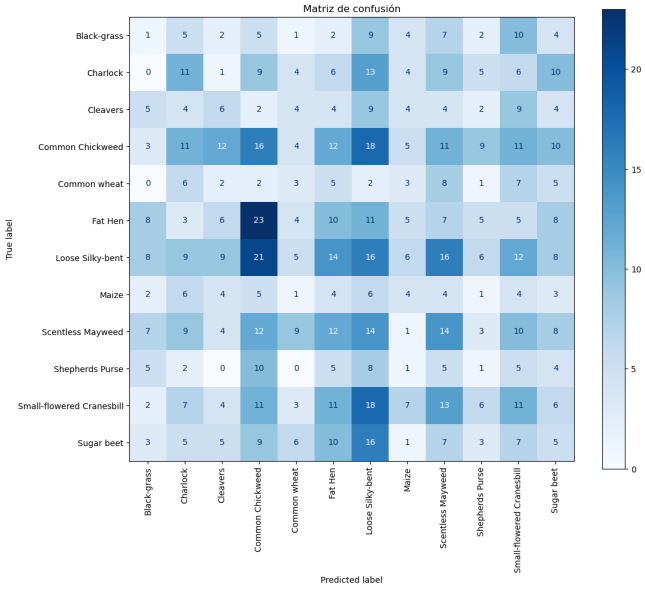

Figure 5. Matriz de confusión

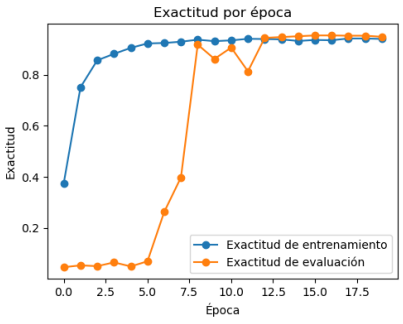

Figure 6. Exactitud por época

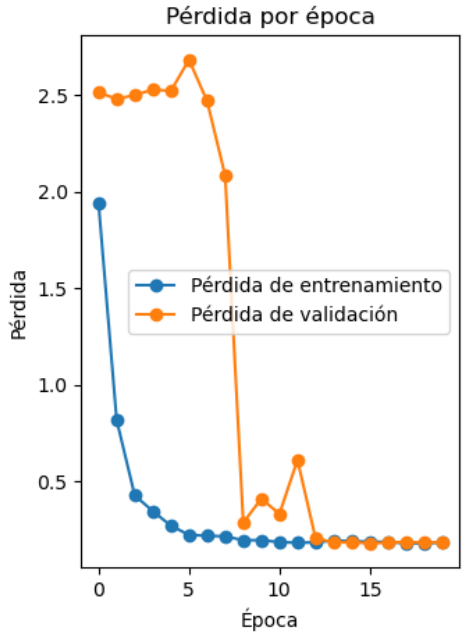

Figure 7. Pérdida por época

	Precisión	Recall	f1-score
Black-grass	0.02	0.02	0.02
Charlock	0.14	0.14	0.14
Cleavers	0.11	0.11	0.11
Common Chickweed	0.13	0.13	0.13
Common wheat	0.07	0.07	0.07
Fat Hen	0.11	0.11	0.11
Loose Silky-bent	0.11	0.12	0.12
Maize	0.09	0.09	0.09
Scentless Mayweed	0.13	0.14	0.13
Shepherds Purse	0.02	0.02	0.02
Small-flowered Cranesbill	0.11	0.11	0.11
Sugar beet	0.07	0.06	0.07

Table 3. Reporte de clasificación

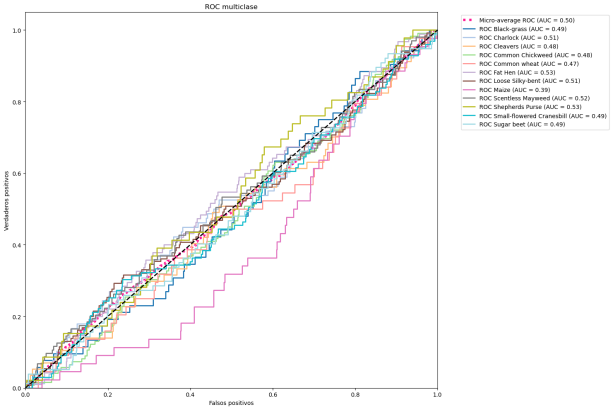

Figure 8. Curvas ROC

Especie	AUC (%)
Black-grass	48.9
Charlock	50.6
Cleavers	48.1
Common Chickweed	48.2
Common wheat	46.7
Fat Hen	52.6
Loose Silky-bent	50.6
Maize	38.7
Scentless Mayweed	51.7
Shepherds Purse	52.9
Small-flowered Cranesbill	48.9
Sugar beet	48.9

Table 4. AUC por especie

6. Discusión

6.1. Análisis de métricas

El modelo alcanzó las siguientes métricas de rendimieento:

- La precisión promediada micro (10.35%) y macro (9.28%) indican un rendimiento bajo en la clasificación de las 12 especies
- El F1-Score ponderado de 10.29% y la exactitud balanceada de 9.32% sugieren que el modelo tiene dificultades para generalizar entre las diferentes clases de plántulas
- Las curvas ROC y sus correspondientes valores AUC (entre 0.387 y 0.529) revelan que el modelo tiene un rendimiento cercano al azar (AUC = 0.5) para la mayoría de las especies
- Las curvas de entrenamiento y validación muestran una convergencia después de aproximadamente 12 épocas, lo que indica que el modelo alcanzó un punto de estabilidad en su aprendizaje

6.2. Interpretación de resultados

- Las especies *Charlock* y *Common Chickweed* mostraron el mejor rendimiento relativo, con precisiones de 14% y 13% respectivamente, sugiriendo que estas especies tienen características más diferenciadas entre sí
- Las especies *Black-grass* y *Shepherds Purse* fueron las más difíciles de clasificar, con solo un 2% de precisión, indicando posibles similitudes entre las especies que confunden al modelo
- El resultado de las curvas ROC indica que la especie *Maize* tiene el peor rendimiento con un AUC de 38.7%, inferior al azar (0.5), lo que podría indicar un problema particular en la clasificación de esta especie
- La matriz de confusión sugiere que las características visuales compartidas entre algunas especies dificultan la clasificación precisa, como es el caso de *Fat Hen* y *Common Chickweed*, además de *Loose Silky-bent* y *Common Chickweed*.

7. Conclusiones

1. **Complejidad de las imágenes:** La clasificación de plántulas representa un reto debido a la similitud entre especies en etapas tempranas de crecimiento. Los resultados sugieren que se requieren características más específicas para mejorar las métricas de la clasificación.
2. **Limitaciones del modelo:** A pesar de usar una arquitectura eficiente y aprendizaje por transferencia, el rendimiento del modelo indica que la arquitectura EfficientNet-B0 podría no ser la más adecuada para este caso específico. El bajo F1-Score ponderado (10.29%) sugiere buscar otras arquitecturas (como EfficientNetB1-B4) si se cuentan con los recursos computacionales suficientes y mejorar las capas adicionales por

medio de padding.

3. **Posibles mejoras:** Los resultados pudieran mejorar mediante: (a) la implementación de técnicas de balanceo, (b) la incorporación de características específicas en el preprocesamiento (como padding), y (c) usar arquitecturas que combinen EfficientNet (B1-B7) para mejorar la resolución de las imágenes o bien, usar pesos de modelos basados en especies de plantas

8. Apéndice

Code 1. Preprocesamiento de imágenes: Configuración del ImageDataGenerator con aumento de datos con rotaciones de 40 grados, zoom del 20%, volteos y división de validación del 20%

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    validation_split=0.2
)

train_generator = train_datagen.
    flow_from_directory(
    train_dir,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.
    flow_from_directory(
    train_dir,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

Code 2. Arquitectura del modelo: Implementación de EfficientNetB0 con pesos de ImageNet, fine-tuning de las últimas 20 capas y agregando capas personalizadas incluyendo GlobalAveragePooling2D y Dropout.

```
base_model = EfficientNetB0(weights='imagenet',
                            include_top=False,
                            input_shape=(IMG_SIZE
    , IMG_SIZE, 3))

# Descongelar las ultimas 20 capas
for layer in base_model.layers[-20:]:
    layer.trainable = True

# Agregar capas personalizadas
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
predictions = Dense(len(categories), activation=
    'softmax')(x)

model = Model(inputs=base_model.input, outputs=
    predictions)

optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Code 3. Configuración de entrenamiento: Implementación de callbacks para early stopping con patience de 5, reducción de learning rate con factor 0.1 y guardado del mejor modelo.

```
# Callbacks
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    patience=3,
    factor=0.1,
    verbose=1
)

checkpoint = ModelCheckpoint(
    'best_model.keras',
    monitor='val_loss',
    save_best_only=True,
    verbose=1
)

# Entrenamiento del modelo
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=20,
    callbacks=[early_stopping, reduce_lr,
    checkpoint]
)
```

Code 4. Evaluación y métricas: Cálculo de métricas de rendimiento incluyendo precisión micro/macro/ponderada, F1-Score ponderado y exactitud balanceada.

```
from sklearn.metrics import precision_score,
    f1_score, balanced_accuracy_score

# Precision promediada
micro_precision = precision_score(Y_true, Y_pred
    , average='micro')
macro_precision = precision_score(Y_true, Y_pred
    , average='macro')
weighted_precision = precision_score(Y_true,
    Y_pred, average='weighted')

# F1-Score y exactitud balanceada
weighted_f1 = f1_score(Y_true, Y_pred, average='
    weighted')
balanced_acc = balanced_accuracy_score(Y_true,
    Y_pred)
```