UNEMI
UNIVERSIDAD ESTATAL DE MILAGRO

FACULTAD DE CIENCIAS E INGENIERÍA
CARRERA DE INGENIERÍA DE SOFTWARE

TEMA:

REDES NEURONALES CONVOLUCIONALES (CNN)

AUTORES:

JUAN DANIEL VICTORES SEMINARIO

KEVIN ANDRES AZUERO RODAS

KENDRYD JONAYKER RODRIGUEZ RONQUILLO

ANTHONY EMILIANO TERAN ARELLANO

ASIGNATURA:

INTELIGENCIA ARTIFICIAL

DOCENTE:

JHONNY DARWIN ORTIZ MATA

FECHA DE ENTREGA:

Domingo, 29 de junio

PERIODO:

abril 2025 a agosto 2025

MILAGRO-ECUADOR

Índice

Análisis Comparativo de Redes Neuronales Convolucionales (CNN) para la Clasificación de Madurez de Bananas

Introducción

El presente proyecto busca impulsar el desarrollo tecnológico en la agricultura ecuatoriana, enfocándose en la industria bananera, uno de los sectores más representativos del país. Dado que Ecuador es el mayor exportador mundial de banano, se destaca la necesidad de implementar soluciones que mejoren su manejo, clasificación y procesamiento.

Por todo lo anteriormente mencionado, se desarrollaron y evaluaron modelos de redes neuronales convolucionales (CNN) capaces de identificar el estado de madurez del banano a partir de imágenes. La clasificación se definió en cuatro categorías: inmaduro, maduro, demasiado maduro y podrido. Para entrenar los modelos se utilizó una dataset con imágenes de esta fruta, además se aplicando tres arquitecturas reconocidas por su efectividad en tareas de visión por computadora: ResNet50, VGG16 y Xception.

Cada modelo fue evaluado en función de su precisión y desempeño general. A partir de esta comparación y se identificó la arquitectura con mejores resultados.

Objetivo General

Desarrollar modelos de predicción basados en redes neuronales convolucionales para clasificar el banano según su estado de maduración, y escoger el mejor en base a métricas cuantitativas, con el objetivo de optimizar los procesos de selección y manejo en la industria bananera ecuatoriana.

Objetivos Específicos

- Preparar y preprocesar un dataset descargado de imágenes representativas de bananos en diferentes estados de maduración.

- Implementar y entrenar modelos de redes neuronales convolucionales para la clasificación automática del estado de maduración del banano.
- Evaluar y comparar el desempeño de los modelos implementados en términos de precisión y eficiencia.
- Seleccionar y proponer el modelo más adecuado para su aplicación práctica en la industria bananera.

Planteamiento del problema

La industria bananera en Ecuador es una de las más importantes a nivel mundial, pero enfrenta desafíos en la clasificación del banano según su grado de maduración. Actualmente, esta clasificación se realiza principalmente de forma manual, lo que conlleva a errores humanos, inconsistencias y un proceso lento que puede afectar la calidad del producto y la eficiencia en la cadena de suministro (Mora, Blomme, & Safari, 2025).

Un mal manejo en la clasificación puede resultar en la entrega de bananos inmaduros, demasiado maduros o incluso en mal estado, lo que genera pérdidas económicas para productores, distribuidores y consumidores (Thiagarajan, Kulkarni, & Jadhav, 2024). Por lo cual, es fundamental implementar una solución tecnológica que permita automatizar y mejorar la precisión en la identificación del estado de maduración del banano.

El uso de inteligencia artificial, especialmente mediante redes neuronales, representa una alternativa prometedora para agilizar y hacer más preciso el proceso de clasificación del banano. Esta tecnología puede fortalecer la competitividad y calidad de la industria bananera ecuatoriana en el mercado global. Por su parte, la evaluación manual del estado de madurez presenta varias limitaciones, entre ellas el elevado consumo de tiempo (Mohamedon, Abd Rahman, Mohamad, & Khalifa, 2021).

Alcance

Este proyecto se enfoca en el desarrollo y evaluación de modelos de predicción basados en redes neuronales convolucionales para clasificar el banano según su estado de maduración, utilizando tres arquitecturas: ResNet50, VGG16 y Xception. Estos modelos permiten identificar el estado del fruto en cuatro categorías: inmaduro, maduro, demasiado maduro y podrido.

El alcance abarca desde la preparación y preprocesamiento del dataset descargado con imágenes representativas del banano, hasta la implementación, entrenamiento y evaluación comparativa de los modelos para seleccionar el que ofrece mejor precisión.

Los modelos creados ayudan a la industria bananera ecuatoriana al agilizar y hacer más confiable la selección del producto mediante análisis visual, aunque se limita al uso de imágenes y no considera aspectos físicos o químicos del fruto.

Marco Referencial

Ecuador es reconocido mundialmente como el mayor exportador de bananos, un producto muy importante para la economía del país y para muchas familias que trabajan en el sector agrícola. Para que el banano llegue en buen estado a los mercados internacionales, es fundamental que se clasifique correctamente según su madurez, ya que esto influye en su calidad y aceptación.

Actualmente, esta clasificación se hace de forma manual, observando el color y aspecto del banano, pero este método puede ser lento y no siempre es preciso, lo que puede causar que algunos frutos se entreguen en un estado inadecuado, afectando tanto a productores como a consumidores (Zahan, 2023).

En los últimos años, la inteligencia artificial y, en especial, las redes neuronales convolucionales han mostrado mucho potencial para ayudar en tareas como esta. Estas tecnologías pueden aprender a reconocer patrones complejos en imágenes y clasificar frutas

automáticamente, lo que puede hacer el proceso más rápido, preciso y confiable (Singh, 2024)

Modelos como ResNet50, VGG16 y Xception son ejemplos de redes que han sido utilizadas con éxito en la clasificación de imágenes agrícolas. Estudios comparativos indican que ResNet50 puede alcanzar precisiones muy altas en la clasificación de frutas, mostrando especial eficacia en la detección y diferenciación de diversas variedades agrícola (Mimma, 2022).

Descripción de la dataset

El dataset de Clasificación de Madurez de Banano contiene un total de 13.478 imágenes de bananos con un tamaño de 416×416 píxeles en diversos estados de maduración. Esta colección está específicamente diseñada para el desarrollo de modelos capaces de clasificar el estado de madurez del fruto, **ver figura 1**.

Para garantizar un entrenamiento y evaluación adecuados, el dataset se encuentra estructurado en tres conjuntos principales:

Conjunto de entrenamiento: Comprende 9.440 imágenes, representando el 70% del dataset total. Este conjunto se utilizó para el entrenamiento directo de los modelos de redes neuronales convolucionales.

Conjunto de validación: Formado por 2.023 imágenes, equivalente al 15% del dataset. Este subconjunto se empleó durante el proceso de entrenamiento para la validación continua y el ajuste de hiperparámetros de los modelos.

Conjunto de prueba: Contiene 2.015 imágenes, constituyendo el 15% restante del dataset. Este conjunto independiente se reservó exclusivamente para la evaluación final de los modelos, permitiendo medir su capacidad de generalización en imágenes no vistas durante el entrenamiento.

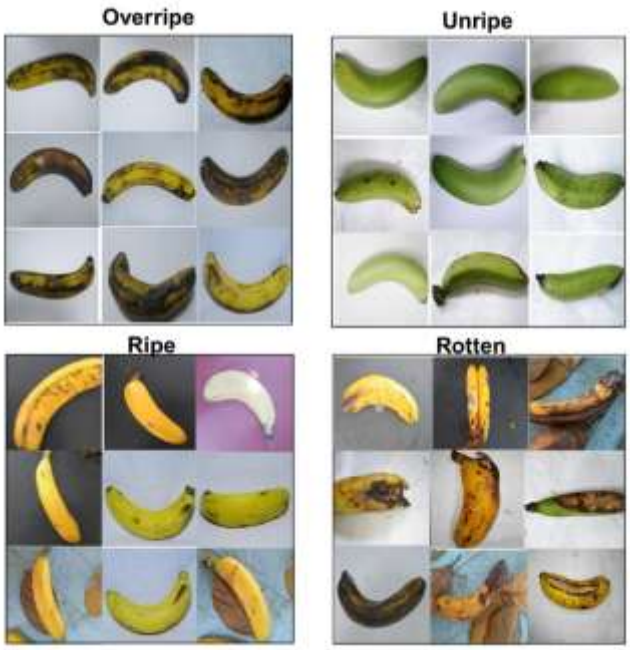

Figura 1 *Banana Ripeness Classification Dataset*

Metodología

Redes Neuronales

La metodología implementada se basa en el uso de redes neuronales convolucionales (CNN) mediante técnicas de transfer learning. Se emplean tres arquitecturas pre-entrenadas en ImageNet: VGG16, ResNet50 y Xception, para aprovechar las características ya aprendidas para la detección de patrones en imágenes (Gabriel, 2021). Estas arquitecturas permiten reutilizar el conocimiento adquirido en la clasificación de millones de imágenes para aplicarlo a un problema específico en este caso la clasificación de madurez de bananas.

Estructura de la Red Neuronal

Capas de Entrada y Salida

Las redes neuronales implementadas adoptan la siguiente arquitectura en común:

- **Entrada:** Imágenes de 224×224 píxeles con excepción de Xception (229×229) con 3 canales de color (RGB).
- **Base congelada:** Modelos pre-entrenados (VGG16, ResNet50, Xception) con pesos de ImageNet congelados.

- **Pooling global:** Capa GlobalAveragePooling2D que reduce la dimensionalidad espacial.
- **Capa densa intermedia:** 128 neuronas con función de activación ReLU.
- **Regularización:** Capa Dropout con probabilidad 0.5 para prevenir sobreajuste.
- **Salida:** 4 neuronas con activación softmax para clasificación multiclase (4 niveles de madurez).

Funciones de Agregación y Activación

Las funciones de activación utilizadas incluyen:

- **ReLU (Rectified Linear Unit):** Aplicada en la capa densa intermedia para introducir no linealidad.
- **Softmax:** Implementada en la capa de salida para generar probabilidades normalizadas para cada clase.
- **GlobalAveragePooling2D:** Función de agregación que promedia espacialmente las características extraídas por las capas convolucionales.

Parámetros de Entrenamiento

Se evaluaron diferentes combinaciones de hiperparámetros *(Tabla 1)* para determinar la configuración óptima del modelo. Para la tasa de aprendizaje, se experimentó con valores de 0.001, 0.0001 y 0.00005, buscando el equilibrio entre velocidad de convergencia y estabilidad del entrenamiento. El batch size se mantuvo constante en 32 muestras, proporcionando un balance adecuado entre eficiencia computacional y estabilidad del gradiente. Se utilizó exclusivamente el optimizador Adam debido a su capacidad de adaptación automática de la tasa de aprendizaje y su robustez en problemas de clasificación de imágenes. El número de épocas varió entre 8, 10 y 12, evaluando el punto óptimo entre

aprendizaje efectivo y prevención del sobreajuste. En todos los experimentos se empleó la función de pérdida Categorical Crossentropy, apropiada para la clasificación multiclase.

Las arquitecturas preentrenadas evaluadas *(Tabla 2)* incluyeron VGG16, ResNet50 y Xception. VGG16 representa la opción más compacta con 14.7 millones de parámetros y 58.9 MB, mientras que ResNet50 es la más robusta con 23.6 millones de parámetros distribuidos en 177 capas. Xception ofrece un equilibrio intermedio con 20.8 millones de parámetros en 135 capas y 88.0 MB de tamaño.

Tabla 1 Configuración de hiperparámetros de entrenamiento

HIPERPARAMETROS	Valores		
Learning rate	0.001	0.0001	0.0005
Batch Size	32		
Optimizer	Adam		
Epochs	8	10	12
Loss function	Categorical crossentropy		

Tabla 2 Detalles de los modelos preentrenados

Model	Parameters	Layers	Size (MB)
VGG16	14.7M	23	58.9
ResNet50	23.6M	177	94.8
Xception	20.8M	135	88.0

Técnicas de Aumento de Datos

Se implementaron técnicas de data augmentation exclusivamente en el conjunto de entrenamiento para incrementar la diversidad del dataset y mejorar la generalización del modelo (Teerath, Alessandra, Rob, & Malika, 2023). Las transformaciones aplicadas incluyen:

- **Rotación:** Hasta 20 grados para simular diferentes orientaciones
- **Desplazamiento:** 20% en ancho y altura para variaciones de posición
- **Corte (shear):** Transformación de corte con factor 0.2
- **Zoom:** Ampliación/reducción hasta 20% del tamaño original
- **Volteo horizontal:** Reflejo horizontal para aumentar variabilidad
- **Preprocesamiento:** Normalización según estándares de VGG16

Matriz de Confusión

La evaluación del modelo se realizó mediante matriz de confusión, una herramienta fundamental para analizar el rendimiento en problemas de clasificación multiclase. Esta matriz permite:

- Visualizar la distribución de predicciones correctas e incorrectas para cada clase
- Identificar patrones de confusión entre las diferentes etapas de madurez
- Calcular métricas adicionales como precisión, recall y F1-score por clase
- Evaluar el equilibrio del modelo en la clasificación de todas las categorías

Librerías de Python

TensorFlow/Keras: Framework principal para la construcción y entrenamiento de redes neuronales, proporcionando herramientas avanzadas para el desarrollo de modelos de deep learning.

NumPy: Librería fundamental para la manipulación eficiente de arrays multidimensionales, optimizando las operaciones matemáticas y el procesamiento de datos numéricos.

Pandas: Herramienta esencial para la gestión, manipulación y análisis de datos estructurados, facilitando la limpieza y transformación de datasets.

Scikit-learn: Biblioteca integral de machine learning que proporciona múltiples métricas de evaluación de modelos, incluyendo recall, F1-score y coeficiente de correlación de Matthews.

Matplotlib/Seaborn: Librerías complementarias para la visualización de datos, generación de gráficos estadísticos y representación de matrices de confusión para el análisis de resultados.

Time: Módulo utilizado para el registro y medición de tiempos de ejecución, permitiendo evaluar el rendimiento computacional de cada modelo implementado.

Justificación del Uso de Redes Neuronales (Xception, ResNet50, VGG16)

En este proyecto de clasificación de bananos, se seleccionaron tres arquitecturas de redes neuronales convolucionales que han demostrado ser especialmente efectivas en tareas de reconocimiento de imágenes: Xception, ResNet50 y VGG16. Cada una aporta características únicas que las hacen adecuadas para la clasificación de bananos en las categorías de bueno, podrido y maduro, desde su capacidad de generalización hasta su eficiencia en el procesamiento de características visuales complejas como textura, color y forma.

Xception

La arquitectura Xception representa un enfoque innovador en el procesamiento de imágenes mediante el uso de convoluciones separables en profundidad. Esta técnica divide el procesamiento en dos etapas: primero analiza la información espacial de la imagen y luego combina esa información de manera eficiente. Esta división del trabajo reduce significativamente el uso de memoria y procesamiento, manteniendo un rendimiento excelente en la clasificación de imágenes.

Además, la estructura del modelo está organizada en tres secciones principales: Entry Flow, Middle Flow y Exit Flow, lo que permite procesar las características de las imágenes de manera gradual y sistemática. Esta organización modular resulta particularmente útil cuando se trabaja con frutas que presentan cambios progresivos en su apariencia física, como es el caso de los bananos durante su proceso de maduración y deterioro. Las convoluciones separables en profundidad reducen la complejidad computacional mientras mantienen la precisión en la clasificación de las diferentes categorías de bananos. (Chollet, 2017)

ResNet50

ResNet50 aborda uno de los problemas más complicados en el entrenamiento de redes neuronales profundas, el desvanecimiento del gradiente que dificulta el aprendizaje en redes con muchas capas. La solución que propone este modelo son las conexiones residuales o "atajos" que permiten que la información fluya más fácilmente a través de toda la red, evitando que se pierda en el proceso de propagación.

Esta arquitectura utiliza bloques residuales que facilitan el entrenamiento de redes profundas de 50 capas, manteniendo la estabilidad del gradiente durante la optimización. Las conexiones residuales permiten que el modelo aprenda características complejas de manera eficiente, siendo especialmente útil para distinguir entre bananos que pueden presentar variaciones significativas dentro de una misma categoría, como bananos buenos con diferentes grados de amarillez o bananos maduros con distintos patrones de manchas.

VGG16

VGG16 se caracteriza por su arquitectura simple pero efectiva, basada en bloques repetitivos de operaciones convolucionales pequeñas (3x3) y operaciones de reducción organizadas de manera regular y predecible. Esta simplicidad arquitectónica facilita su implementación, comprensión y ajuste según las necesidades específicas del proyecto.

Su estructura uniforme la convierte en una excelente base para el aprendizaje por transferencia, donde se pueden aprovechar conocimientos previamente aprendidos en otras tareas de clasificación de imágenes. A pesar de ser una arquitectura más antigua en comparación con Xception y ResNet50, VGG16 mantiene su relevancia debido a su confiabilidad y estabilidad durante el entrenamiento (Simonyan, 2014).

Descripción de la Arquitectura de las Redes Neuronales Propuestas

El modelo Xception emplea convoluciones separables en profundidad para procesar imágenes de 299 x 299 x 3. En el Entry Flow aplica convoluciones estándar con 32 y 64 filtros seguidas de MaxPooling como se ilustra en la figura 2. Posteriormente utiliza convoluciones separables que optimizan la extracción de características iniciales. Esta arquitectura mejora la eficiencia computacional respecto a convoluciones tradicionales. Las convoluciones separables son la característica distintiva que define esta red. El Entry Flow establece las bases para el procesamiento de características complejas.

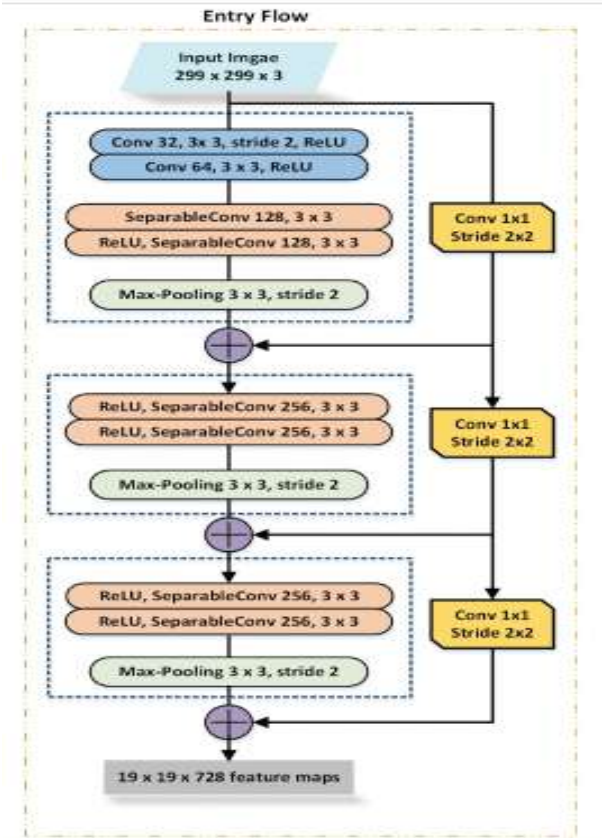

Figura 2 *Representación de las primeras capas del modelo Xception (entry flow)*

En el modelo Xception se estructura en Entry Flow, Middle Flow y Exit Flow como etapas principales. El Middle Flow procesa mapas de 19x19 con 728 canales mediante convoluciones separables y ReLU. Esta etapa permite análisis profundo manteniendo eficiencia computacional gracias a las convoluciones separables. El Exit Flow incluye GlobalAveragePooling antes de la clasificación final con Softmax. Las conexiones entre etapas optimizan el flujo de gradientes durante el entrenamiento. La arquitectura completa maximiza el rendimiento con menor costo computacional como se muestra en la figura 3.

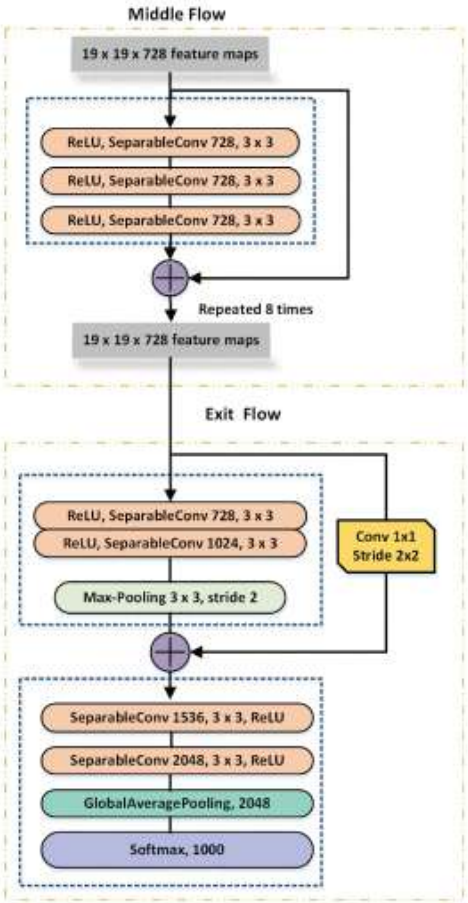

Figura 3 *Arquitectura general del modelo Xception (middle flow)*

RestNet50 es la arquitectura mostrada en la figura 3, introduce conexiones residuales que solucionan el problema de degradación de gradientes en redes profundas. Inicia con convolución 7×7 de 64 filtros y MaxPooling como bloque de entrada. Contiene tres bloques residuales principales con convoluciones 1×1, 3×3 y 1×1 en configuraciones específicas. Las conexiones de salto permiten que la información fluya directamente entre bloques evitando la pérdida de gradiente. Los filtros aumentan progresivamente: 64-256, 256-1024, y 512-2048 en cada bloque respectivamente. Esta arquitectura permite entrenar redes significativamente más profundas manteniendo el rendimiento

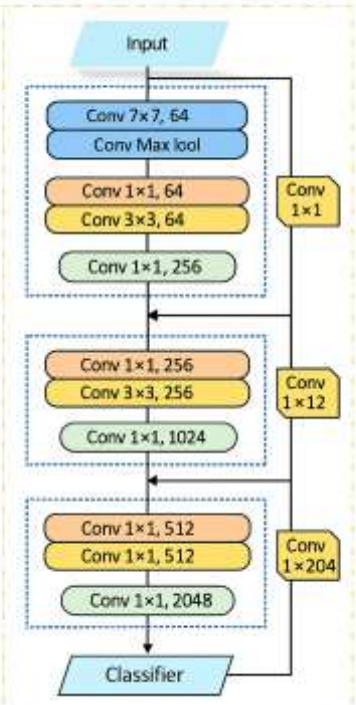

Figura 4 *Arquitectura de red neuronal convolucional Resnet50*

.

VGG16: Utiliza únicamente convoluciones 3×3 y MaxPooling organizadas en cinco bloques principales. La arquitectura mostrada en la figura 5, progresa sistemáticamente duplicando filtros: 64, 128, 256, 512, 512 en cada bloque respectivo. Cada bloque reduce dimensiones espaciales mediante MaxPooling mientras aumenta la profundidad de características. Las 13 capas convolucionales extraen características jerárquicas de complejidad creciente. Finaliza con dos capas densas de 4096 neuronas que integran todas las características extraídas.

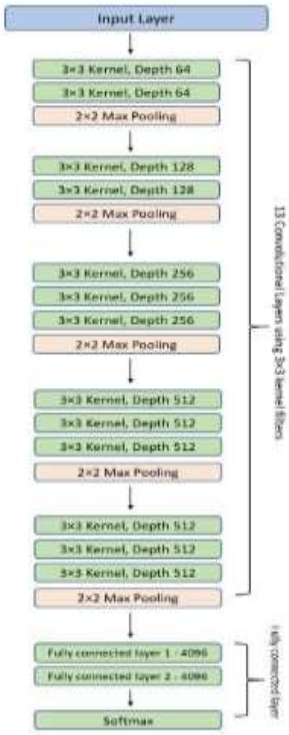

Figura 5 *Arquitectura de la red neuronal convolucional VGG-16*

Desarrollo del programa en Python y comentario de código de programación

```
base_dir = './Banana Ripeness Classification Dataset V2'
print(os.listdir(base_dir))
```

Se carga el dataset

```
lr ="00005"
Et = "10"
l_rate = 0.00005
Epoch_test = 10
```

Estas variables separan texto y números según su uso: lr y Et sirven para nombrar archivos sin errores (como evitar puntos), y l_rate y Epoch_test se usan como valores numéricos para entrenar el modelo correctamente.

```
def save_test_accuracy(experiment_name, test_accuracy,
filename='test_accuracy_log.csv'):

    new_entry = pd.DataFrame([{   'experiment': experiment_name,
'test_accuracy': test_accuracy }])
    if os.path.exists(filename):

            df_existing = pd.read_csv(filename)

            # Eliminar cualquier fila con el mismo experimento

            df_existing = df_existing[df_existing['experiment'] !=
experiment_name]

          # Agregar nueva entrada

          df_combined = pd.concat([df_existing, new_entry],
ignore_index=True)

  else:

    df_combined = new_entry

  df_combined.to_csv(filename, index=False)

  print(f"Test accuracy for '{experiment_name}' saved to
'{filename}'.")
```

Esta función guarda la precisión de prueba (test_accuracy) de un experimento en un archivo CSV llamado test_accuracy_log.csv. Si el archivo ya existe, primero lo lee, elimina cualquier fila que tenga el mismo nombre de experimento (experiment_name) para evitar duplicados, y luego añade la nueva entrada con la precisión actual. Si el archivo no existe, crea uno nuevo con esa entrada. Finalmente, guarda el DataFrame actualizado en el archivo CSV y muestra un mensaje confirmando el guardado.

```
class MetricsLogger(Callback):
```

Clase para poder registrar métricas personalizadas durante el entrenamiento de un modelo en Keras. Calcula y guarda métricas como accuracy, pérdida, recall, F1-score, MCC y tiempo por época, tanto para entrenamiento como validación, y permite guardarlas en un archivo CSV al finalizar.

```
def init(self, validation_data, experiment_name=None):
      super(MetricsLogger, self).init()
      self.validation_data = validation_data
      self.history = []
      self.experiment_name = experiment_name
      self.epoch_start_time = None
```

Inicializa los datos de validación y el nombre del modelo en caso de ser proporcionado, así mismo una lista vacía para guardar el historial de métricas y una variable para registrar el tiempo de inicio de cada época.

```
def on_epoch_begin(self, epoch, logs=None):

            self.epoch_start_time = time.time()
```

Función que se ejecuta al iniciar una época y registra el tiempo de inicio para calcular su duración.

```
def on_epoch_end(self, epoch, logs=None):

     epoch_duration = time.time() - self.epoch_start_time

     val_data, val_labels = self.validation_data

     val_preds = self.model.predict(val_data, verbose=0)

     val_preds_classes = np.argmax(val_preds, axis=1)

     val_true_classes = np.argmax(val_labels, axis=1)

     recall = recall_score(val_true_classes, val_preds_classes,
average='macro')
            f1 = f1_score(val_true_classes, val_preds_classes,
average='macro')
            mcc = matthews_corrcoef(val_true_classes,
val_preds_classes)

            logs = logs or {}

            # Entrenamiento
            train_accuracy = logs.get('accuracy')
            train_loss = logs.get('loss')

          # Validación
            val_accuracy = logs.get('val_accuracy')
            val_loss = logs.get('val_loss')

             self.history.append({ 'experiment':
self.experiment_name if self.experiment_name else 'default',
'epoch': epoch + 1, 'accuracy': train_accuracy, 'loss':
train_loss, 'val_accuracy': val_accuracy, 'val_loss': val_loss,
'recall': recall, 'f1_score': f1, 'mcc': mcc, 'epoch_time':
round(epoch_duration, 2)})
```

Esta función se ejecuta al final de cada época del entrenamiento. Calcula cuánto duró la época, obtiene los datos y etiquetas de validación, y genera las predicciones del modelo. Con estas, calcula métricas como recall, F1 y MCC. También toma del historial las métricas de accuracy y pérdida del entrenamiento y validación. Al final, guarda toda esta información, duración, métricas y datos del experimento en un historial para su análisis posterior.

```
def save_to_csv(self, filename='metrics_log.csv'):

         new_df = pd.DataFrame(self.history)

    if os.path.exists(filename):
        existing_df = pd.read_csv(filename)

        # Eliminar filas con experimentos ya existentes
        existing_df =
existing_df[~existing_df['experiment'].isin(new_df['experiment'])]
        # Combinar con los nuevos datos
        combined_df = pd.concat([existing_df, new_df],
ignore_index=True)
    else:
        combined_df = new_df

    combined_df.to_csv(filename, index=False)
    print(f"Metrics saved to '{filename}' with {len(new_df)} new
rows.")
```

Función para poder guardar el historial de métricas almacenado en un archivo CSV, evitando duplicar experimentos existentes y combinando datos nuevos con los previos si el archivo ya existe.

```
image_size = ( #Altura , #Anchura )
```

Tamaño al que se redimensionan las imágenes: VGG16 y ResNet50 usan 224x224 píxeles; Xception usa 299x299 píxeles.

```
batch_size = 32

train_datagen =
ImageDataGenerator(preprocessing_function=preprocess_input,
rotation_range=20, width_shift_range=0.2, height_shift_range=0.2,
shear_range=0.2, zoom_range=0.2, horizontal_flip=True)

valid_test_datagen =
ImageDataGenerator(preprocessing_function=preprocess_input,)

train_generator =
train_datagen.flow_from_directory( os.path.join(base_dir,
'train'), target_size=image_size, batch_size=batch_size,
class_mode='categorical' )

valid_generator =
valid_test_datagen.flow_from_directory( os.path.join(base_dir,
'valid'), target_size=image_size, batch_size=batch_size,
class_mode='categorical', )

test_generator =
valid_test_datagen.flow_from_directory( os.path.join(base_dir,
'test'), target_size=image_size, batch_size=batch_size,
class_mode='categorical', shuffle=False # Importante para la
matriz de confusión )
```

Se prepara los datos de entrenamiento, validación y prueba para un modelo de clasificación de imágenes. Se aplica aumentos como rotación, desplazamiento, zoom y volteo horizontal solo a las imágenes de entrenamiento para mejorar el rendimiento del modelo. Las imágenes de validación y prueba se preprocesan sin aumentos. Luego, se crean generadores que leen imágenes desde carpetas (train, valid, test), las redimensionan, las agrupan en lotes de 32 y las codifican como categorías. El conjunto de prueba no se mezcla para mantener el orden al evaluar resultados.

```
base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))

base_model.trainable = False # Congelar pesos de VGG16

x = base_model.output x = GlobalAveragePooling2D()(x)

x = Dense(128, activation='relu')(x)

x = Dropout(0.5)(x)

predictions = Dense(4, activation='softmax')(x)

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

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

model.summary()
```

Se crea un modelo preentrenado con VGG16 para clasificar imágenes en 4 clases. Se carga VGG16 sin su capa final, se congelan sus pesos y se le agregan capas personalizadas: global average pooling, una densa de 128 neuronas con ReLU, un dropout del 50% y una capa de salida con 4 neuronas y activación softmax. Luego se compila el modelo con Adam, categorical_crossentropy y accuracy, y se muestra su resumen.

```
base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))

base_model.trainable = False

x = base_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(128, activation='relu')(x)

x = Dropout(0.5)(x)

predictions = Dense(4, activation='softmax')(x) # 4 clases como
antes
```

```
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer=Adam(learning_rate= l_rate),
loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

Se crea un modelo preentrenado con ResNet50 para clasificar imágenes en 4 clases. Se carga ResNet50 sin su capa final, se congelan sus pesos y se le agregan capas personalizadas: global average pooling, una densa de 128 neuronas con ReLU, un dropout del 50% y una capa de salida con 4 neuronas y activación softmax. Luego se compila el modelo con Adam, categorical_crossentropy y accuracy, y se muestra su resumen.

```
base_model = Xception(weights='imagenet', include_top=False,
input_shape=(299, 299, 3))

base_model.trainable = False

x = base_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(128, activation='relu')(x)

x = Dropout(0.5)(x)

predictions = Dense(4, activation='softmax')(x) # 4 clases como
antes

model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer=Adam(learning_rate=l_rate),
loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()
```

Se crea un modelo preentrenado con Xception para clasificar imágenes en 4 clases. Se carga Xception sin su capa final, se congelan sus pesos y se le agregan capas personalizadas: global average pooling, una densa de 128 neuronas con ReLU, un dropout del 50% y una capa de salida con 4 neuronas y activación softmax. Luego se compila el modelo con Adam, categorical_crossentropy y accuracy, y se muestra su resumen.

```
def generator_to_arrays(generator):
        X_list, y_list = [], []
        for i in range(len(generator)):
              x, y = generator[i]
              X_list.append(x) # <- aquí
              y_list.append(y)
         return np.concatenate(X_list), np.concatenate(y_list)
X_val, y_val = generator_to_arrays(valid_generator)
metrics_logger = MetricsLogger(validation_data=(X_val, y_val),
experiment_name=f'resnet50_{l_rate}_{Epoch_test}')
```

Esta función convierte un generador de datos en dos arrays de NumPy: uno para las entradas (X) y otro para las salidas (y). La función generator_to_arrays recorre el generador, toma cada lote (x, y) y guarda x en una lista X_list y y en y_list. Luego, une ambas listas en arrays y los devuelve. Después, esos arrays (X_val y y_val) se usan en un objeto MetricsLogger para guardar métricas de validación durante el entrenamiento, usando un nombre que incluye la tasa de aprendizaje y la cantidad de épocas

```
with tf.device('/GPU:0'):
       history = model.fit( train_generator, epochs=Epoch_test,
validation_data=valid_generator, callbacks=[metrics_logger] )
            metrics_logger.save_to_csv('resultados_modelos.csv')
```

entrena un modelo usando la GPU (/GPU:0) para aprovechar un mayor rendimiento. Utiliza datos de entrenamiento y validación durante un número de épocas definido por Epoch_test, y registra métricas mediante el callback metrics_logger. Al finalizar, guarda esas métricas en un archivo CSV llamado 'resultados_modelos.csv'.

```
    test_accuracy = accuracy_score(test_generator.classes,
y_pred) experiment_name = f'nombreModelo_{l_rate}_{Epoch_test}'

    save_test_accuracy(experiment_name, test_accuracy)
```

Se calcula la precisión del modelo sobre el conjunto de datos de prueba y guarda ese valor junto con el nombre del modelo (VGG16, xception o ResNet50), que incluye el modelo, la tasa de aprendizaje y el número de épocas.

```
    model.save(f'modelos/nombreModelo_{lr}_{Et}.h5')
```

Guarda el modelo localmente, con su respectivo nombre (VGG16, xception o ResNet50)

En términos generales el código desarrollado sigue un proceso ordenado en cuatro etapas principales. Primero se configuran los parámetros y se preparan los datos del dataset de bananas. Luego se crean las funciones necesarias para registrar y guardar las métricas de cada experimento. En la tercera etapa se construyen los tres modelos (VGG16, ResNet50 y Xception) usando redes preentrenadas y adaptándolas para clasificar los cuatro niveles de madurez. Finalmente, cada modelo se entrena, evalúa y guarda automáticamente junto con sus resultados, permitiendo comparar el rendimiento entre las diferentes arquitecturas de manera sistemática.

Análisis de resultados

Análisis de Rendimiento de Modelos CNN

En la Figura 6**,** se observa la evolución del accuracy de los diferentes modelos a lo largo de 12 épocas de entrenamiento. La tendencia general muestra un comportamiento ascendente y estable, con fluctuaciones menores que no comprometen el aprendizaje. Es notable el incremento dramático del accuracy entre las épocas 1 y 2, comportamiento típico en redes neuronales profundas donde el modelo ajusta rápidamente sus pesos desde valores aleatorios iniciales hacia patrones más representativos, capturando las características más evidentes de las clases. A partir de la época 7, los modelos alcanzan una fase de estabilización con rendimiento más consistente.

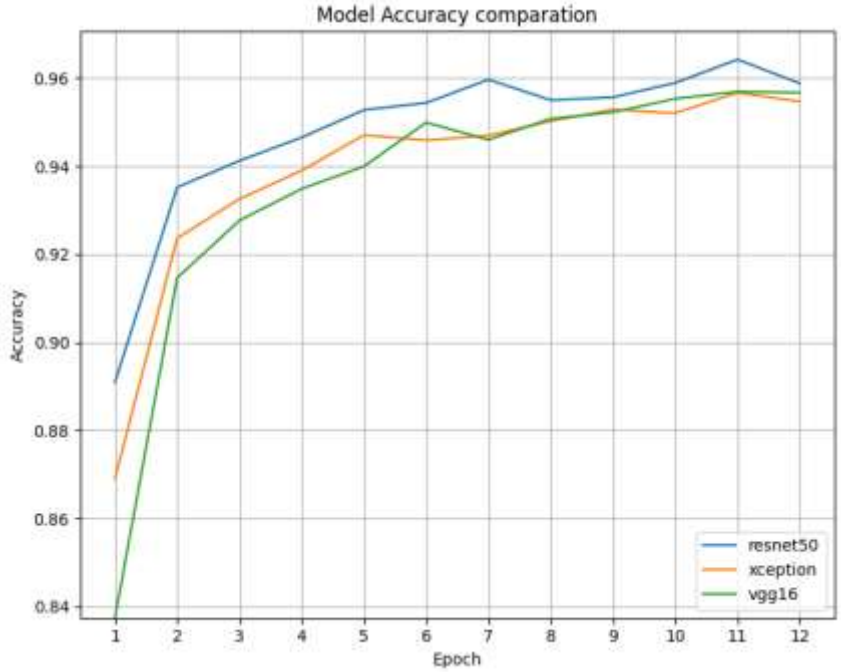

Figura 6 *Gráfico de precisión del entrenamiento para diferentes redes*

Aunque la diferencia no es muy notoria, el modelo que obtuvo mayor accuracy al final de la época época 12 fue ResNet50 con un porcentaje cercano al 96%. Sin embargo, este resultado genera dudas debido al claro descenso en la precisión observado entre las épocas 7 a 9, además de que alcanzó un pico superior al 96% pero descendió en la época final.

Por otro lado, Xception, aunque logró una precisión decente, resultó ser el menor rendimiento obtenido al finalizar la época 12. Este modelo mostró mayor estabilidad que ResNet50, pero aún presentó descensos entre las épocas 5-6 y 9-10, manteniendo un rendimiento estable en el resto del entrenamiento.

VGG16 obtuvo una precisión intermedia, superior a Xception pero inferior a ResNet50. Este modelo presentó la curva más estable, con únicamente cierto desbalance entre las épocas 6 y 7. A partir de ese punto, se observó una tendencia ascendente que se estabilizó entre las épocas 11 y 12, demostrando un comportamiento más consistente y confiable durante el proceso de entrenamiento.

En la Figura 7 se presenta el comportamiento de la función de pérdida (loss) para los diferentes modelos de clasificación durante 12 épocas de entrenamiento, evidenciando qué tan bien se ajustan las predicciones del modelo a las etiquetas reales. Esta métrica constituye un parámetro fundamental para evaluar la efectividad del proceso de aprendizaje, donde su disminución progresiva señala una mejora en la capacidad predictiva del sistema.

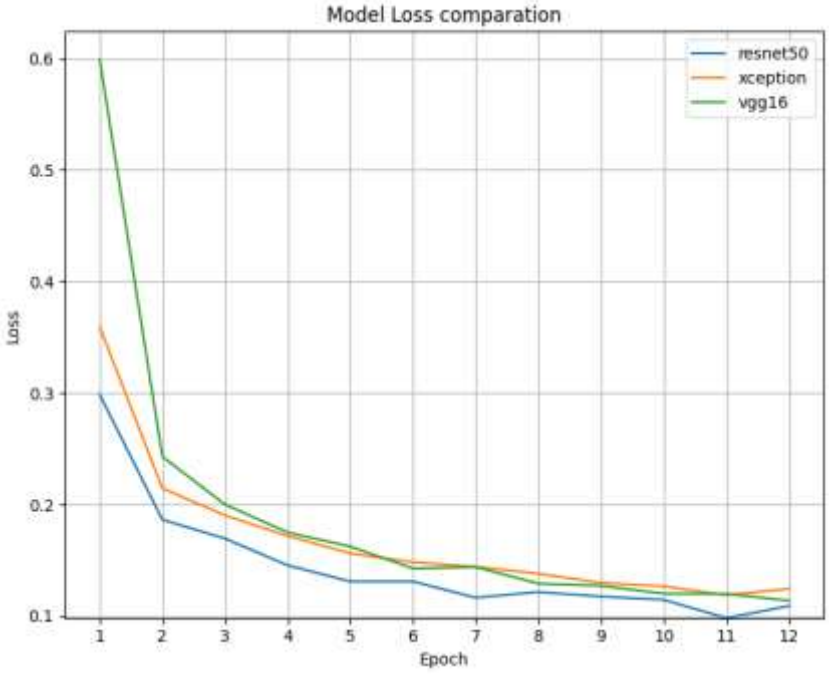

Figura 7 *Diagrama de pérdida durante el entrenamiento de diferentes redes neuronales*

Durante las primeras épocas, los tres modelos experimentan una disminución notoria de la pérdida, ver figura 7**,** alcanzando posteriormente un comportamiento más progresivo y gradual en su descenso. Todos los modelos lograron valores muy bajos de pérdida, lo cual demuestra una excelente capacidad de convergencia y ajuste a los patrones de clasificación de los datos.

ResNet50 presenta un incremento evidente entre las épocas 11 y 12, lo cual podría indicar el inicio de sobreajuste si se continuara el entrenamiento, sugiriendo que alcanzó su punto óptimo en épocas anteriores. Xception muestra un comportamiento similar, aunque menos pronunciado que ResNet50, con cierto aumento en estas mismas épocas finales. Por su parte, VGG16 exhibió irregularidades menores entre las épocas 6 y 8, para posteriormente mantener una tendencia decreciente en las épocas subsiguientes.

Es importante destacar que estas variaciones son mínimas en magnitud, por lo que no representarían un riesgo inmediato para el rendimiento general de los modelos, manteniéndose dentro de rangos aceptables para la clasificación.

Resultados de Entrenamiento

La Tabla 3 presenta una evaluación exhaustiva del desempeño de modelos de redes neuronales implementando el optimizador Adam con un batch size fijo de 32. El estudio contempló un diseño experimental completo que abarcó 3 arquitecturas distintas de redes neuronales, cada una entrenada con 3 configuraciones diferentes de épocas y 3 valores distintos de learning rate, resultando en un total de 27 entrenamientos sistemáticos.

Tabla 3 *Resultados obtenidos con distintas redes neuronales convolucionales (CNN)*

Model	Bacth size	Epoch	Learning Rate	Training time	Validation Accuracy	Test Accuracy	Parameters
VGG16	32	8	0.00005	609s	93,97%	93,25%	14.7M
			0.0001	607s	95,50%	94,93%	
			0.001	605s	97,78%	98,26%	
		10	0.00005	836s	95,55%	95,18%	
			0.0001	761s	97,08%	97,32%	
			0.001	768s	97,92%	97,56%	
		12	0.00005	902s	96,44%	95,73%	
			0.0001	917s	97,18%	96,77%	
			0.001	982s	97,83%	98,51%	
Xception	32	8	0.00005	1135s	94,66%	92,25%	20,8 M
			0.0001	1120s	96,69%	95,38%	
			0.001	1119s	97,78%	97,46%	
		10	0.00005	1498s	95,16%	93,69%	
			0.0001	1412s	96,64%	95,33%	
			0.001	1405s	97,83%	97,66%	
		12	0.00005	1782s	96,05%	93,84%	
			0.0001	1702s	97,18%	95,88%	
			0.001	1679s	97,83%	97,27%	
Resnet50	32	8	0.00005	736s	96,19%	96,67%	23.6M
			0.0001	721s	97,38%	97,32%	
			0.001	606s	97,73%	97,81%	
		10	0.00005	933s	96,54%	97,02%	
			0.0001	909s	97,43%	97,66%	
			0.001	896s	96,44%	97,36%	
		12	0.00005	1084s	96,00%	95,78%	
			0.0001	1077s	97,28%	97,66%	
			0.001	977s	97,73%	97,41%	

En cuanto a los tiempos de entrenamiento, la arquitectura Xception fue la que requirió mayor tiempo computacional, con un promedio de 1,428 segundos. Le siguió la arquitectura ResNet50 con una duración promedio de 882 segundos. Por su parte, VGG16 demostró ser la más eficiente en términos temporales, registrando un promedio de 776 segundos de entrenamiento.

En términos de precisión, se identificó la combinación óptima de hiperparámetros para cada arquitectura evaluada. Los resultados muestran que todas las configuraciones de mejor rendimiento utilizaron una tasa de aprendizaje de 0.001, variando únicamente en el número de épocas según la arquitectura.

VGG16 se posicionó como la arquitectura con mejor desempeño general, alcanzando su configuración óptima con 12 épocas y una tasa de aprendizaje de 0.001. Esta combinación logró una precisión de validación del 97.83% y una precisión de prueba superior del 98.51%, representando el mejor resultado obtenido en el conjunto de prueba.

Xception, con su configuración óptima de 10 épocas y tasa de aprendizaje de 0.001, obtuvo una precisión de validación del 97.83% (igual a VGG16) y una precisión de prueba del 97.66%. Aunque ligeramente inferior en el conjunto de prueba, mantiene un rendimiento competitivo.

ResNet50 alcanzó su mejor desempeño con 8 épocas y tasa de aprendizaje de 0.001, registrando una precisión de validación del 97.73% y una precisión de prueba del 97.81%. Si bien presenta la menor precisión de validación, su rendimiento en el conjunto de prueba supera al de Xception.

Para obtener una evaluación más completa del desempeño de los modelos, se analizaron métricas adicionales incluyendo precisión, recall, F1-score y MCC (Matthews Correlation Coefficient), cuyos resultados se presentan en la Tabla 4**.** Esta evaluación se realizó utilizando los modelos con las combinaciones óptimas de parámetros de entrenamiento identificadas para cada arquitectura.

Tabla 4 *Métricas de evaluación para los modelos de clasificación*

Model	Accuracy	Precision	Recall	F1 Score	MCC
RESNET50	0.9781	0.9790	0.9781	0.9782	0.9710
VGG16	0.9851	0.9851	0.9851	0.9851	0.9801
XCEPTION	0.9766	0.9767	0.9766	0.9766	0.9688

VGG16 confirmó su superioridad al mostrar valores estables alrededor del 98.51% en accuracy, precisión y recall, con solo el MCC presentando una ligera diferencia al alcanzar 98.01%. Esta consistencia demuestra un rendimiento equilibrado y confiable en todas las métricas evaluadas.

ResNet50 y Xception mantuvieron un rendimiento competitivo con valores estables en torno al 97-98% para todas las métricas, siendo sus valores de MCC los que presentaron las mayores variaciones respecto a las otras métricas. Ambas arquitecturas demostraron ser opciones viables con un desempeño sólido y consistente.

Como análisis final, se aplicó **fine-tuning** a los modelos para evaluar posibles mejoras en su rendimiento. En la Figura 8 se presenta una comparativa entre los modelos con y sin fine-tuning aplicado.

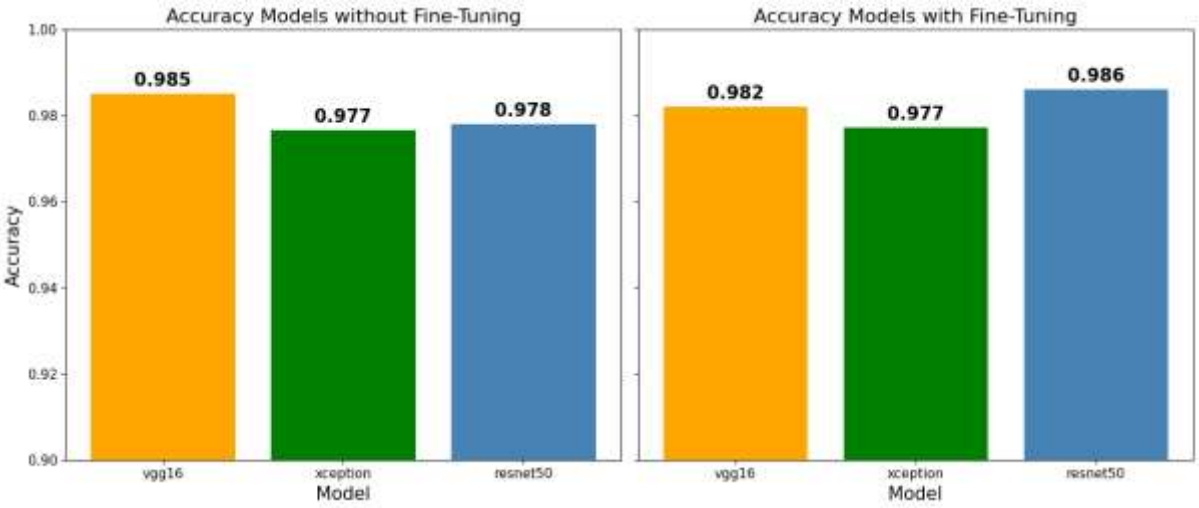

Figura 8 *Rendimiento de modelos antes y después del fine-tuning*

Los resultados muestran comportamientos diferenciados según la arquitectura. **ResNet50** fue la única arquitectura que experimentó una mejora significativa, incrementando su accuracy de 97.8% a 98.6%, lo que representa una ganancia de 0.8 puntos porcentuales.

Por el contrario, **VGG16** mostró un efecto negativo del fine-tuning, experimentando una reducción en su precisión respecto al modelo original. Este resultado sugiere que VGG16 ya había alcanzado un punto óptimo de rendimiento en su configuración inicial.

Xception prácticamente no presentó cambios destacables tras la aplicación de fine-tuning, manteniendo un rendimiento similar al modelo base, lo que indica estabilidad en su arquitectura pero sin beneficios adicionales por esta técnica.

Para validar si ResNet50 con fine-tuning se posicionaba como el mejor modelo, se evaluaron nuevamente todas las métricas de rendimiento, para los modelos FT, cuyos resultados se muestran en la Tabla 5.

Tabla 5 Métricas de evaluación para los modelos con Fine-Tuning

Model FT	Accuracy	Precision	Recall	F1 Score	MCC
RESNET50	0.9861	0.9864	0.9861	0.9860	0.9815
VGG16	0.9821	0.9829	0.9821	0.9821	0.9715
XCEPTION	0.9771	0.9773	0.9771	0.9771	0.9695

Los resultados revelan que **ResNet50 y VGG16** obtienen los valores más altos entre las tres arquitecturas evaluadas. ResNet50 con fine-tuning presenta un rendimiento ligeramente superior, consolidándose como una opción competitiva. Sin embargo, al comparar con los valores originales de la Tabla 4, se observa que VGG16 experimentó una reducción no solo en accuracy, sino también en el resto de las métricas evaluadas.

Por su parte, **Xception** mostró ligeras mejoras respecto a su versión sin fine-tuning, aunque las diferencias son mínimas y no representan cambios significativos en su rendimiento genera.

Matrices de Confusión

Para evaluar el rendimiento de los modelos de manera más específica, se generaron matrices de confusión para los mejores modelos obtenidos, los cuales fueron probados con un conjunto de prueba compuesto por 2,015 imágenes distribuidas de la siguiente manera: 479 imágenes de frutos sobremaduros (overripe), 520 de frutos maduros (ripe), 541 de frutos en descomposición (rotten) y 475 de frutos verdes o inmaduros (unripe). Estas matrices permitieron analizar no solo la precisión general de cada modelo, sino también su capacidad para distinguir correctamente entre las diferentes etapas de maduración del producto.

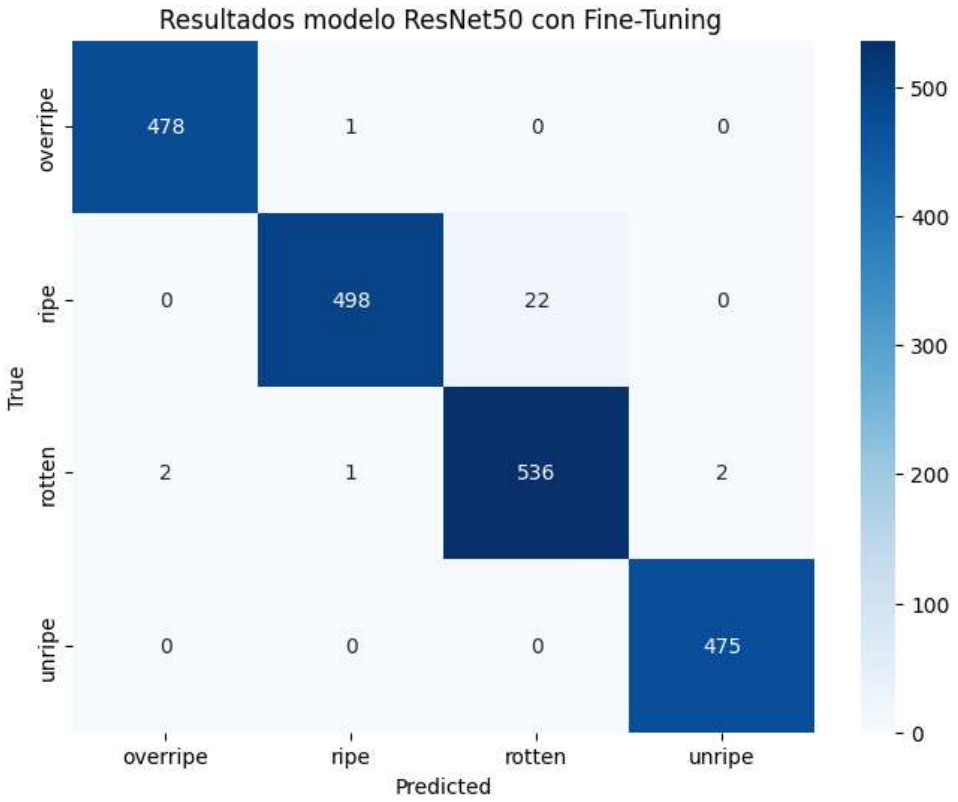

Figura 9 *Matriz de confusión de ResNet50 con fine-tuning*

ResNet50 con Fine-Tuning presenta una elevada precisión global, con un rendimiento perfecto en la clase "unripe", sin errores de clasificación. En la clase "overripe", apenas se observa 1 error, clasificado como "ripe", mientras que en "rotten" se registran solo 4 errores (2 como "overripe" y 2 como "unripe"). La clase con mayor confusión fue "ripe",

con 22 muestras clasificadas erróneamente como "rotten", aunque mantuvo una alta tasa de aciertos. Estos resultados, visibles en la figura 9**,** consolidan a ResNet50 con Fine-Tuning como un modelo sólido y confiable para esta tarea de clasificación.

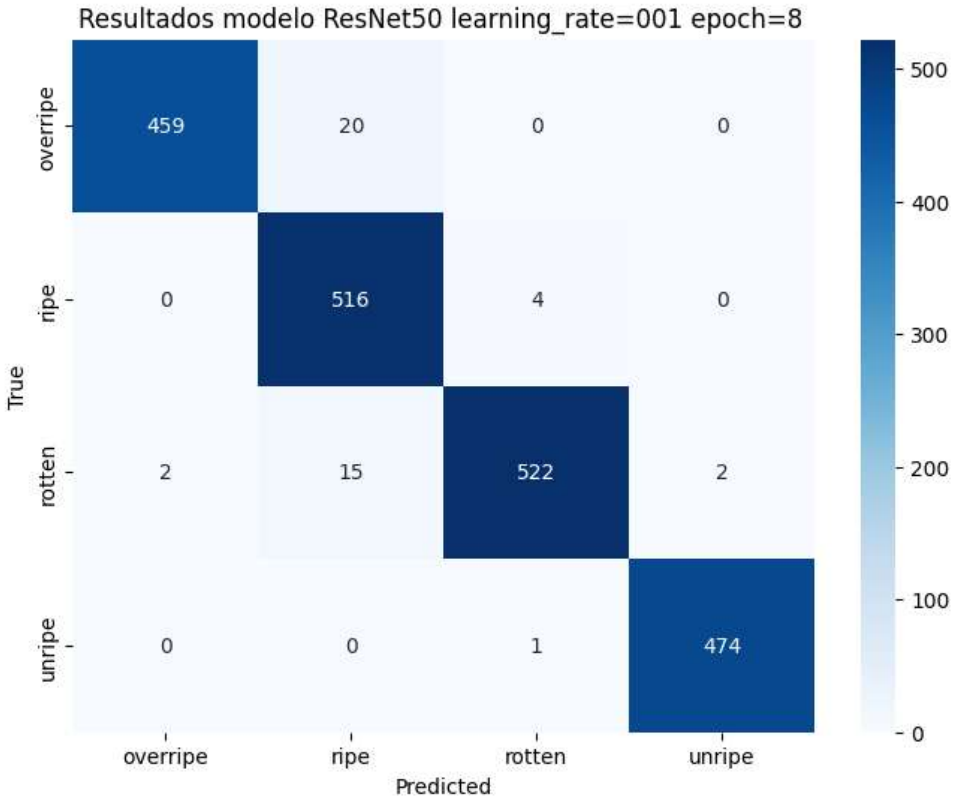

Figura 10 *Matriz de confusión de ResNet50*

Para el modelo ResNet50 sin Fine-Tuning, ver figura 10**.** También se obtuvo una matriz con buenos resultados, aunque inferiores a los de su contraparte con Fine-Tuning. La clase "overripe" presentó la mayor cantidad de errores (20), principalmente confundida con "ripe". En "ripe", solo se cometieron 4 errores, mientras que "rotten" tuvo un total de 19 errores, 15 de ellos confundidos con "ripe". La clase con mejor desempeño fue nuevamente "unripe", con únicamente 1 error de predicción.

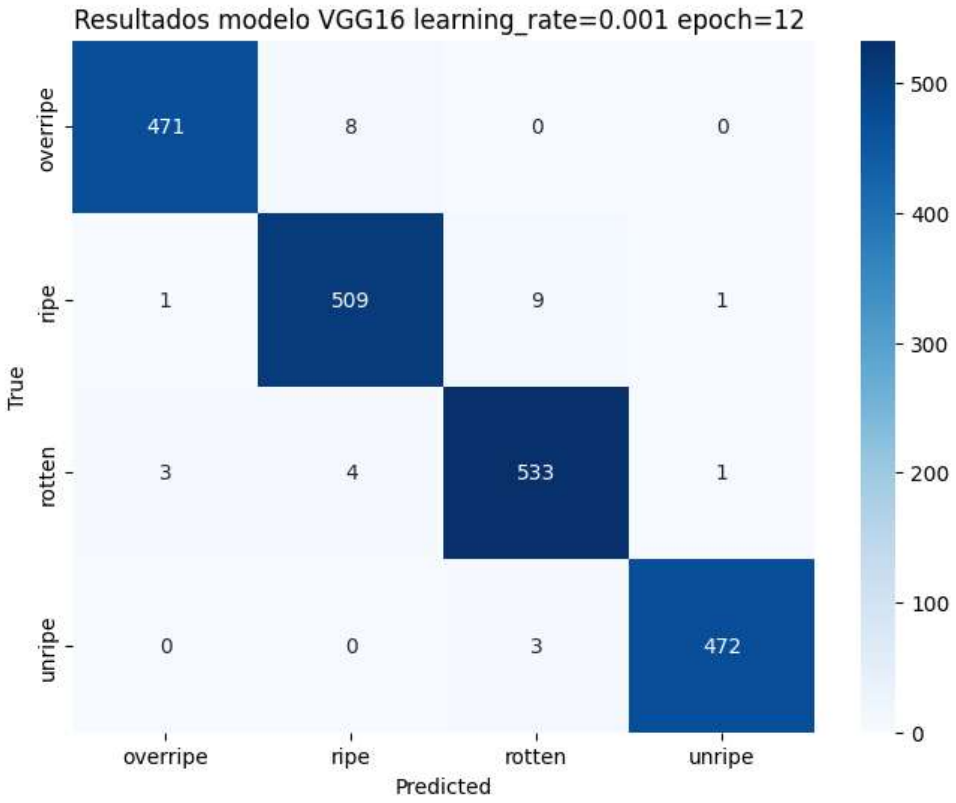

Figura 11 *Matriz de confusión de VGG16*

El segundo modelo con mejores resultados fue VGG16 sin fine tuning, ver figura 11**,** con una precisión casi tan alta como la de ResNet50. Aunque presenta una mayor cantidad total de errores, estos están más equilibradamente distribuidos entre las clases. En "overripe", se registraron solo 8 errores, todos confundidos con "ripe". Para la clase "ripe", hubo 11 errores: 9 con "rotten", y 1 con "unripe" y "overripe", respectivamente. En "rotten", se detectaron 8 errores, distribuidos en 4 con "ripe", 3 con "overripe" y 1 con "unripe". Al igual que en los modelos anteriores, "unripe" se mantiene como la clase con mejor desempeño, con solo 3 errores, todos clasificados como "rotten".

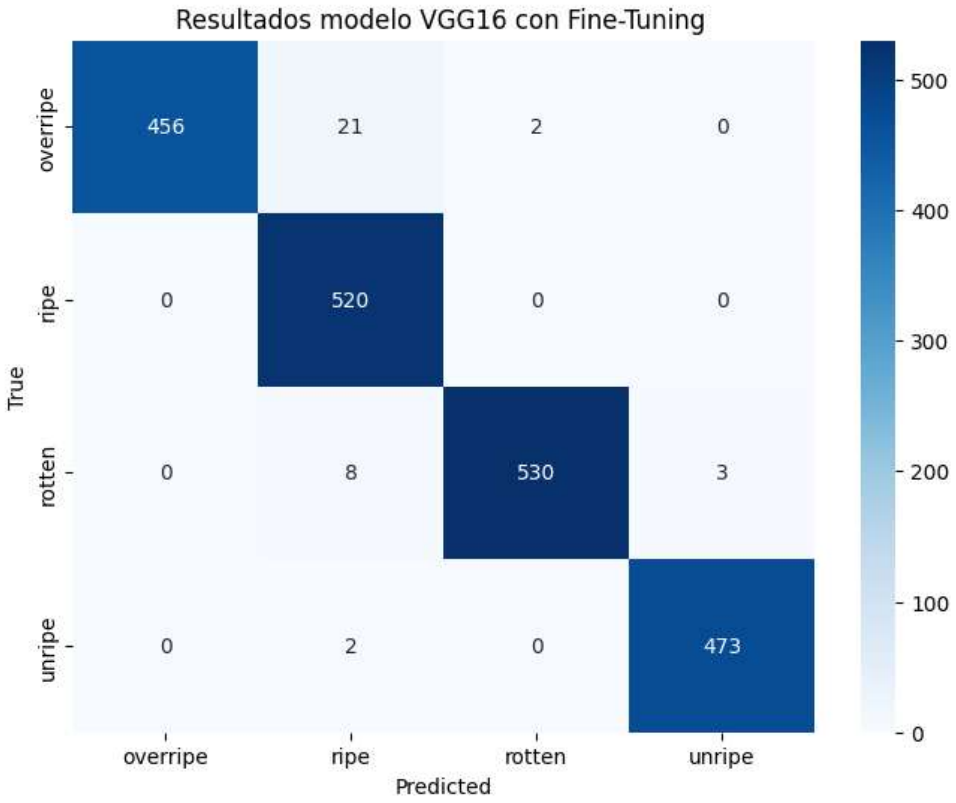

Figura 12 *Matriz de confusión de VGG16 con fine-tuning*

El modelo VGG16 con Fine-Tuning, ver Figura 12**.** Presenta una precisión aceptable, aunque inferior en términos generales a su versión sin Fine-Tuning. Llamativamente, es el modelo que obtiene los mejores resultados en la clase "ripe", que suele concentrar la mayor cantidad de errores en otros modelos, logrando 520 aciertos y 0 errores. Sin embargo, presenta una cantidad considerable de errores en las demás clases: "overripe" con 456 aciertos y 23 errores, "rotten" con 530 aciertos y 11 errores, y "unripe" con 473 aciertos y 2 errores.

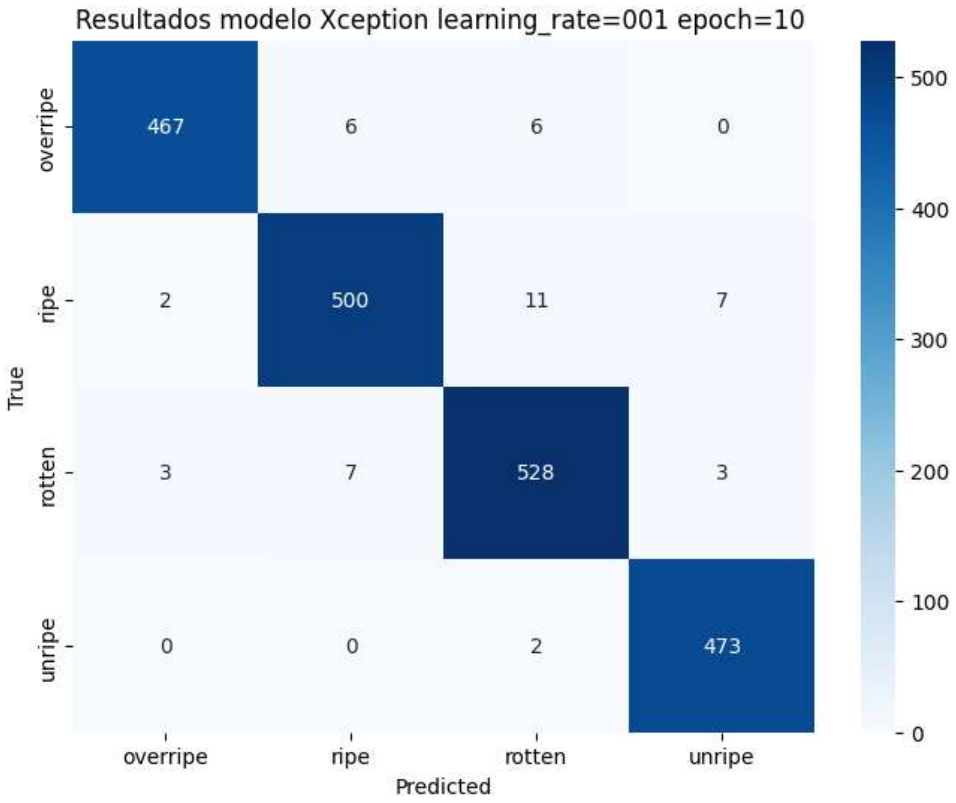

Figura 13 *Matriz de confusión de Xception*

El modelo Xception sin fine-tuning obtuvo el rendimiento más bajo en comparación con su versión optimizada y el resto de los modelos evaluados; sin embargo, esto no indica un mal desempeño, ya que alcanzó una precisión general cerca al 97% de aciertos, ver figura 13. Los resultados por clase mostraron que la categoría "overripe" logró 467 predicciones correctas con 12 errores, "ripe" obtuvo 500 aciertos y 20 errores, "rotten" presentó 528 predicciones acertadas con 13 errores, y finalmente "unripe" destacó con 473 aciertos y únicamente 2 errores, siendo esta última la clase con mejor desempeño en términos de precisión.

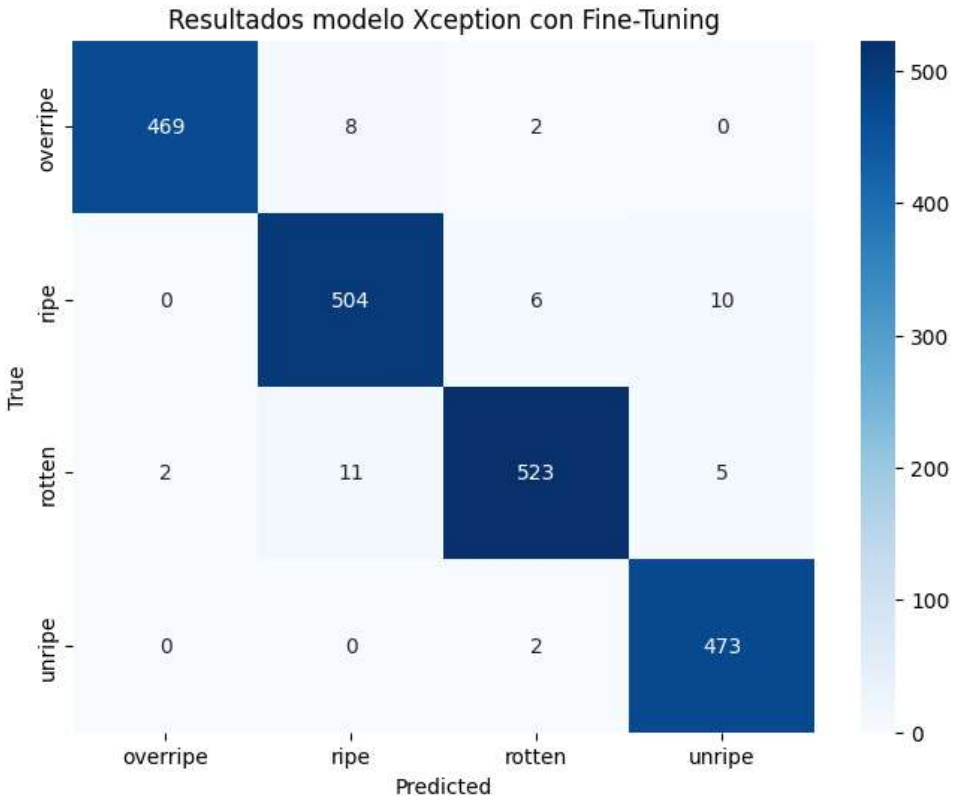

Figura 14 *Matriz de confusión de Xception con fine-tuning*

El modelo Xception con fine-tuning se posicionó en el penúltimo lugar general entre los modelos evaluados, ver figura 14. Aunque mostró una precisión aceptable, no alcanzó los niveles de rendimiento de ResNet50 y VGG16.

En el análisis por clases obtuvo los siguientes resultados: overripe (469 aciertos, 10 errores), ripe (504 aciertos, 16 errores), rotten (523 aciertos, 18 errores) y unripe (473 aciertos, 2 errores). La clase rotten evidenció la mayor imprecisión, mientras que unripe mostró el mejor desempeño.

Conclusiones

- VGG16 demostró ser la arquitectura más efectiva para la clasificación de estados de maduración del banano, alcanzando la mayor precisión en el conjunto de prueba (98.51%) con su configuración óptima de 12 épocas y learning rate de 0.001. Esta arquitectura destacó no solo por su rendimiento superior, sino también por presentar la curva de entrenamiento más estable, con mínimas fluctuaciones y un comportamiento ascendente consistente a partir de la época 7. Adicionalmente, VGG16 mostró la mayor eficiencia computacional con un tiempo promedio de entrenamiento de 776 segundos, siendo significativamente más rápida que Xception (1,428 segundos) y ResNet50 (882 segundos), lo que la convierte en una solución óptima tanto en precisión como en recursos computacionales.
- El fine-tuning mostró efectos variables dependiendo de la arquitectura empleada, evidenciando que no todas las CNN se ben efician de esta técnica en igual medida. ResNet50 fue la única arquitectura que experimentó una mejora significativa con fine-tuning, incrementando su accuracy de 97.8% a 98.6% y consolidándose como el modelo con mejor rendimiento post-optimización. En contraste, VGG16 experimentó una degradación en su rendimiento tras la aplicación de fine-tuning, sugiriendo que ya había alcanzado su punto óptimo en la configuración inicial. Xception mantuvo un rendimiento prácticamente inalterado, indicando estabilidad, pero sin beneficios adicionales. Estos resultados subrayan la importancia de evaluar individualmente el impacto del fine-tuning para cada arquitectura específica.
- Todas las arquitecturas evaluadas demostraron una capacidad excepcional para identificar correctamente la clase "unripe" (bananos verdes o inmaduros), presentando consistentemente las menores tasas de error en esta categoría. Este

patrón se mantuvo constante tanto en modelos con fine-tuning como sin él, sugiriendo que las características visuales distintivas de los bananos inmaduros son fácilmente capturadas por las CNN independientemente de su arquitectura. La mayor dificultad se concentró en la diferenciación entre las clases "ripe" y "rotten", donde se registraron las confusiones más frecuentes, indicando que la transición entre estados de maduración avanzada presenta desafíos visuales más complejos para los modelos de clasificación automática

- El análisis reveló que todas las configuraciones de mejor rendimiento convergieron hacia un learning rate de 0.001, independientemente de la arquitectura utilizada, estableciendo este valor como óptimo para la tarea de clasificación de bananos. Adicionalmente, se observó que las arquitecturas más complejas no necesariamente requieren mayor número de épocas para alcanzar su rendimiento óptimo: ResNet50 alcanzó su mejor desempeño con solo 8 épocas, mientras que VGG16 y Xception necesitaron 12 y 10 épocas respectivamente. Los análisis de la función de pérdida evidenciaron signos tempranos de sobreajuste en ResNet50 y Xception hacia las épocas finales, sugiriendo que el entrenamiento prolongado puede ser contraproducente y que la implementación de técnicas de early stopping podría optimizar aún más el rendimiento de estos modelos.

Recomendaciones

- Se recomienda estudiar a fondo el dataset antes de seleccionar la arquitectura, ya que este puede ser el causante directo del bajo rendimiento de un modelo. Es fundamental revisar la cantidad de imágenes para cada conjunto, analizar qué imágenes contiene y verificar si existen diferencias significativas entre ellas. Esta evaluación determinará cómo se tratarán los datos y cómo se construirá la arquitectura del proyecto.
- Las copias de seguridad o versionado es vital para proyectos de este tipo, ya que los tiempos de entrenamiento de modelos y los recursos utilizados son considerables. La pérdida de un registro o archivo puede ocasionar que se restablezca por completo el entrenamiento de un modelo, por lo que es recomendable crear código que facilite el registro de los datos de entrenamiento y prueba.
- Se recomienda utilizar plataformas especializadas en la creación de modelos CNN como Paperspace, ya que, al no contar con hardware suficientemente potente, la culminación de proyectos de este tipo resulta prácticamente imposible. Estas plataformas proporcionan los recursos computacionales necesarios para el desarrollo eficiente de modelos de deep learning.
- Es vital entender las bases del machine learning y conceptos fundamentales como overfitting y underfitting para comprender la problemática que enfrenta el proyecto y cómo solucionarla. Asimismo, dominar conceptos como épocas, tasa de aprendizaje y optimizadores contribuye significativamente a mejorar el rendimiento del modelo y tomar decisiones informadas durante el proceso de desarrollo.

Bibliografía

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (págs. 1800–1807). Honolulu, HI: IEEE. doi:10.1109/CVPR.2017.195

Gabriel, C. (16 de Junio de 2021). *Transfer Learning with VGG16 and Keras*. Obtenido de Towards data science: https://towardsdatascience.com/transfer-learning-with-vgg16-and-keras-50ea161580b4/

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity mappings in deep residual networks. *Computer Vision – ECCV 2016* (págs. 630–645). Amsterdam: Springer. doi:10.1007/978-3-319-46493-0_38

Mimma, S. A. (2022). Fruits Classification and Detection Application Using Deep Learning. *Scientific Programming, 2022*, 4194874. doi:10.1155/2022/4194874

Mohamedon, M., Abd Rahman, F., Mohamad, S., & Khalifa, O. O. (2021). Clasificación de la madurez del plátano mediante una aplicación móvil basada en visión artificial. *8.ª Conferencia Internacional de Ingeniería Informática y de la Comunicación (ICCCE)*, (págs. 335-338). Kuala Lumpur. doi:10.1109/ICCCE50029.2021.9467225

Mora, J., Blomme, G., & Safari, N. (2025). Digital framework for georeferenced multiplatform surveillance of banana wilt using human in the loop AI and YOLO foundation models. *Scientific Reports, 15*, 3491. doi:10.1038/s41598-025-87588-2

Simonyan, K. &. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 1-14. Obtenido de https://arxiv.org/abs/1409.1556

Singh, A. (2024). Deep learning and computer vision in plant disease detection: a comprehensive review of techniques, models, and trends in precision agriculture. *Artificial Intelligence Review, 57*(1), 1-70. doi:10.1007/s10462-024-11100-x

Teerath, K., Alessandra, M., Rob, B., & Malika, B. (2023). Image Data Augmentation Approaches: A Comprehensive Survey and Future directions. *arxiv*. doi:https://doi.org/10.48550/arXiv.2301.02830

Thiagarajan, J., Kulkarni, S., & Jadhav, S. (2024). Analysis of banana plant health using machine learning techniques. *Scientific Reports, 14*, 15041. doi:10.1038/s41598-024-63930-y

Zahan, N. (2023). Deep learning based intelligent identification system for ripening stages of banana. *Postharvest Biology and Technology, 201*, 112370. doi:10.1016/j.postharvbio.2023.112370