

Apéndice C. Procedimientos complementarios para el diseño y entrenamiento del modelo CNN y YOLO

C.1 Preparación de los conjuntos de entrenamiento

Una vez finalizadas las etapas de adquisición y preprocesamiento descritas en los apéndices A y B, se procedió a la preparación de los conjuntos de entrenamiento utilizados durante el desarrollo de la red neuronal convolucional. Para ello, se emplearon las imágenes RGB previamente organizadas en las categorías Buenas y Malas, realizando su carga automática mediante TensorFlow y aplicando estrategias de aumento de datos con el fin de mejorar la capacidad de generalización del modelo.

C.1.1 Carga del dataset RGB

Una vez conformado el conjunto de datos RGB y realizado su preprocesamiento previamente descrito en los apéndices A y B, se procedió a la preparación del conjunto de datos utilizado para el entrenamiento de la red neuronal convolucional. Para ello, se empleó TensorFlow, realizando la carga automática de las imágenes organizadas en las categorías Buenas y Malas, previamente redimensionadas a una resolución de 100×100 píxeles.

Las muestras fueron agrupadas en lotes de tamaño fijo y posteriormente divididas en conjuntos de entrenamiento y validación. Para garantizar la reproducibilidad de los experimentos, se empleó una semilla fija durante el proceso de carga y aleatorización de las imágenes.

Código C.1. *Fragmento del script empleado para la carga y división del conjunto de datos.*

```
1 IMG_SIZE = (100, 100)
2 BATCH_SIZE = 32
3
4 dataset = tf.keras.utils.image_dataset_from_directory(
5     DATASET_PATH,
6     labels='inferred',
7     label_mode='int',
8     class_names=['Buenas', 'Malas'],
9     image_size=IMG_SIZE,
10    batch_size=BATCH_SIZE,
11    shuffle=True,
12    seed=42
13 )
14
15 dataset_size = tf.data.experimental.cardinality(dataset).numpy()
16 train_size = int(0.70 * dataset_size)
17
18 train_ds_raw = dataset.take(train_size)
19 val_ds_raw = dataset.skip(train_size)
```

Posteriormente, el conjunto de datos fue dividido en aproximadamente un 70 % de las muestras para entrenamiento y un 30 % para validación. Esta separación permitió evaluar el comportamiento del modelo sobre datos no utilizados durante el proceso de aprendizaje, facilitando el análisis de su capacidad de generalización.

Tabla C.1. *Parámetros utilizados para la carga y división del conjunto de datos RGB.*

Parámetro	Valor
Resolución de entrada	100 × 100 píxeles
Número de clases	2
Clases	Buenas y Malas
Tamaño de lote	32
Proporción entrenamiento	70 %
Proporción validación	30 %
Framework empleado	TensorFlow

C.1.2 Data augmentation

Con el propósito de mejorar la capacidad de generalización de la red neuronal convolucional y reducir el riesgo de sobreajuste, se implementó una estrategia de aumento de datos (data augmentation) sobre las imágenes RGB del conjunto de entrenamiento. Esta técnica permitió generar variaciones aleatorias de las muestras originales sin alterar la clase a la que pertenecían, incrementando la diversidad de ejemplos observados por el modelo durante el proceso de aprendizaje.

Las transformaciones fueron implementadas mediante las herramientas proporcionadas por TensorFlow y aplicadas dinámicamente durante el entrenamiento, evitando la necesidad de almacenar nuevas imágenes en disco. Entre las operaciones empleadas se incluyeron rotaciones aleatorias, aumentos y reducciones de escala, así como inversiones horizontales y verticales de las imágenes.

Código C.2 *Fragmento del pipeline para el Aumento de datos.*

```
1 data_augmentation = tf.keras.Sequential([
2     tf.keras.layers.RandomRotation(0.08),
3     tf.keras.layers.RandomZoom(0.3),
4     tf.keras.layers.RandomFlip("horizontal"),
5     tf.keras.layers.RandomFlip("vertical"),
6 ])
7
```

Las transformaciones aplicadas permitieron que el modelo fuera expuesto a diferentes orientaciones y escalas de las mandarinas, favoreciendo la extracción de características más robustas y aumentando la capacidad de adaptación frente a variaciones presentes durante las etapas de inferencia. Debido a que las mandarinas pueden presentarse con distintas posiciones sobre la

banda transportadora, estas operaciones contribuyeron a mejorar la invariancia espacial del modelo y a incrementar la diversidad efectiva del conjunto de entrenamiento.

Figura C.1 *Ejemplos de imágenes RGB obtenidas mediante las técnicas de data augmentation empleadas en el entrenamiento del modelo.*



Tabla C.2. *Operaciones de aumento de datos implementadas durante el entrenamiento de la red CNN.*

Operación	Parámetro empleado	Propósito
Rotación aleatoria	± 0.08	Variar la orientación del fruto.
Zoom aleatorio	0.3	Simular cambios de escala.
Inversión horizontal	Activada	Aumentar la diversidad espacial.
Inversión vertical	Activada	Mejorar la robustez frente a diferentes posiciones.

C.1.3 Preparación y optimización del pipeline de datos

Con el fin de preparar las muestras para el entrenamiento de la red neuronal convolucional, se implementó un pipeline de procesamiento utilizando TensorFlow. En esta etapa, las imágenes fueron convertidas al formato de punto flotante y normalizadas en el intervalo [0,1], facilitando el

proceso de aprendizaje del modelo y favoreciendo la estabilidad numérica durante el entrenamiento.

Adicionalmente, las operaciones de aumento de datos fueron aplicadas únicamente sobre el conjunto de entrenamiento, mientras que las imágenes pertenecientes al conjunto de validación conservaron sus características originales. Finalmente, se implementaron mecanismos de paralelización y precarga de datos mediante la función *prefetch*, con el propósito de optimizar el flujo de información hacia la red neuronal y reducir los tiempos de procesamiento.

Código C.3. *Fragmento del pipeline de procesamiento implementado para la normalización y optimización del flujo de datos.*

```
1 def preprocess(img, label, training=True):
2     img = tf.cast(img, tf.float32) / 255.0
3     if training:
4         img = data_augmentation(img)
5     return img, label
6
7 train_ds = train_ds_raw.map(
8     lambda x, y: preprocess(x, y, training=True),
9     num_parallel_calls=tf.data.AUTOTUNE
10 ).prefetch(tf.data.AUTOTUNE)
11
12 val_ds = val_ds_raw.map(
13     lambda x, y: preprocess(x, y, training=False),
14     num_parallel_calls=tf.data.AUTOTUNE
15 ).prefetch(tf.data.AUTOTUNE)
```

La normalización de las imágenes permitió escalar los valores de intensidad de los píxeles entre 0 y 1, favoreciendo una convergencia más estable durante el entrenamiento. Asimismo, la utilización de la función *AUTOTUNE* permitió que TensorFlow ajustara automáticamente los recursos de procesamiento empleados en la lectura y preparación de los datos, mientras que la

operación prefetch facilitó la carga anticipada de los lotes de imágenes, mejorando la eficiencia del proceso de entrenamiento.

Figura C.2. Esquema del pipeline de procesamiento implementado para la preparación y optimización de los datos utilizados durante el entrenamiento de la red neuronal convolucional.



Tabla C.3. Operaciones implementadas en el pipeline de procesamiento de datos.

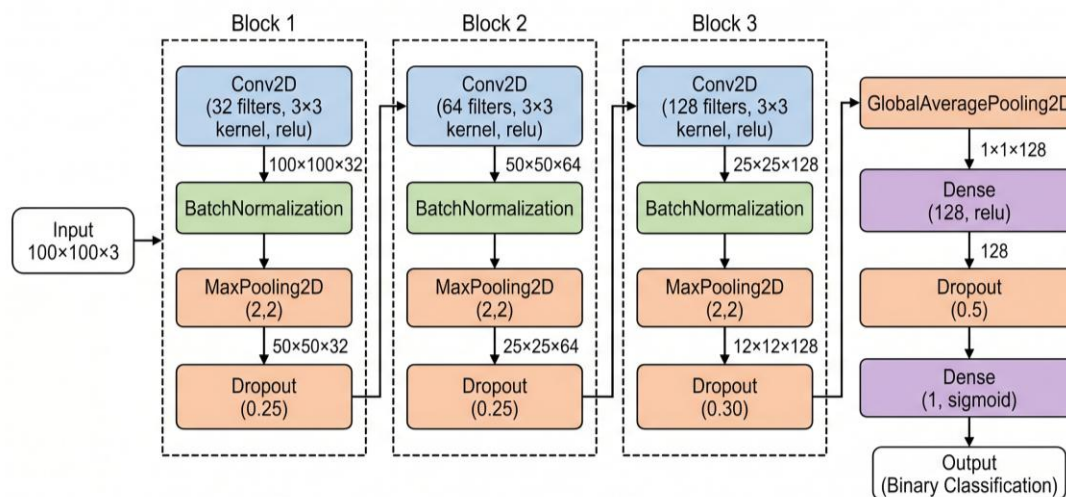
Operación	Descripción
Conversión a float32	Adaptación del formato numérico de las imágenes.
Normalización	Escalado de intensidades al intervalo [0,1].
Data augmentation	Aplicado únicamente al conjunto de entrenamiento.
map()	Aplicación del pipeline sobre los datos
AUTOTUNE	Ajuste automático de los recursos de procesamiento.
prefetch()	Precarga de lotes para mejorar la eficiencia.

C.1.4 Arquitectura de la red neuronal convolucional

Con el propósito de realizar la clasificación automática de las muestras RGB en las categorías apta y no apta, se diseñó una red neuronal convolucional utilizando TensorFlow y Keras. La arquitectura recibió como entrada imágenes RGB de 100 × 100 píxeles y estuvo conformada por tres bloques convolucionales con una progresión de 32, 64 y 128 filtros, respectivamente. Cada bloque integró capas Conv2D, BatchNormalization y MaxPooling2D, complementadas con Dropout para reducir el riesgo de sobreajuste durante el entrenamiento.

Como se observa en la Figura C.3, el incremento progresivo del número de filtros permitió obtener representaciones de mayor nivel de abstracción a medida que la información avanzaba a través de la red, mientras que las operaciones de agrupamiento redujeron gradualmente las dimensiones espaciales de los mapas de características.

Figura C.3. Arquitectura de la red neuronal convolucional desarrollada para la clasificación binaria de las muestras RGB.



Después de los bloques convolucionales, se utilizó una capa GlobalAveragePooling2D para condensar los mapas de características en un vector de 128 elementos. Posteriormente, este vector fue procesado mediante una capa densa de 128 neuronas con activación ReLU y una capa Dropout de 0,50. Finalmente, se empleó una neurona de salida con función de activación sigmoid para obtener la probabilidad asociada a la clasificación binaria de cada muestra.

Las capas convolucionales y la capa densa intermedia utilizaron la función de activación ReLU, mientras que la capa de salida empleó la función sigmoid. Esta última genera valores

comprendidos entre 0 y 1, permitiendo obtener la probabilidad utilizada para asignar cada muestra a una de las dos categorías consideradas.

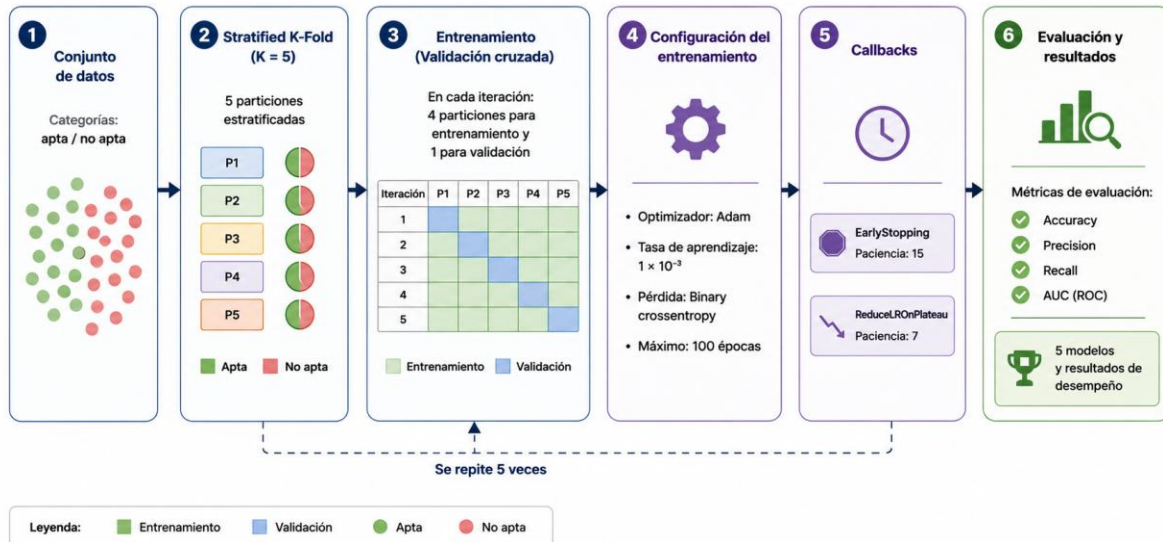
C.1.5 Validación cruzada y configuración del entrenamiento

Para evaluar la estabilidad de aprendizaje y reducir la dependencia de una única partición de los datos, se implementó un procedimiento de validación cruzada estratificada mediante Stratified K-fold con cinco particiones. Esta estrategia permitió conservar aproximadamente la proporción entre las categorías apta y no apta en cada subconjunto de entrenamiento y validación. En cada iteración se inicializo y entreno un nuevo modelo desde cero, utilizando cuatro particiones para el entrenamiento y la restante para la validación. El procedimiento se repitió hasta utilizar cada partición una vez como conjunto de validación, obteniendo así cinco modelos y sus correspondientes resultados de desempeño.

Para el entrenamiento, el modelo fue compilado utilizando el optimizador Adam con una tasa de aprendizaje inicial de $1e-3$ y la función de perdida binary crossentropy. El proceso se configuro para un máximo de 100 épocas y fue supervisado mediante las métricas de exactitud (accuracy), precision (precisión), sensibilidad (recall) y área bajo la curva ROC (AUC).

Con el propósito de controlar la duración del entrenamiento y reducir el riesgo de sobreajuste, se implementaron los callbacks EarlyStopping y ReduceLROnPlateau. El primero permitió detener el entrenamiento cuando la métrica de validación dejó de presentar mejoras durante 15 épocas consecutivas, mientras que el segundo redujo automáticamente la tasa de aprendizaje después de siete épocas sin mejora.

Figura C.4. Esquema del procedimiento de validación cruzada estratificada de cinco particiones utilizado durante el entrenamiento del modelo RGB.



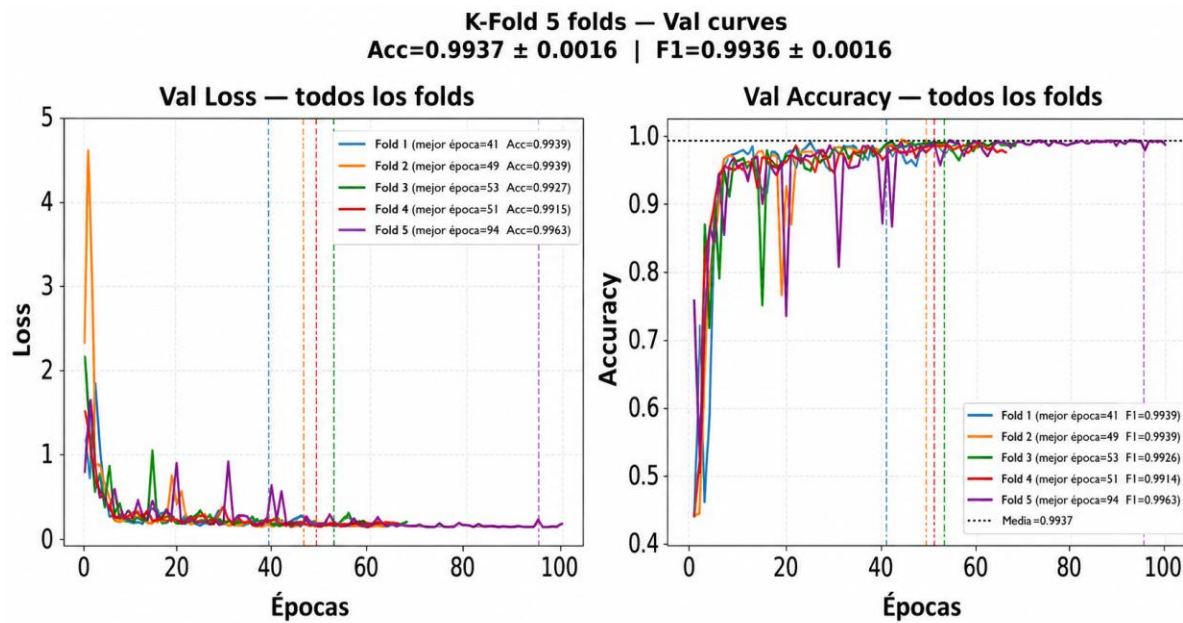
C.1.6 Entrenamiento y almacenamiento del modelo final

Una vez finalizado el procedimiento de validación cruzada, se utilizaron las mejores épocas obtenidas en los cinco folds como referencia para establecer la duración del entrenamiento del modelo final. A partir de los valores registrados en cada partición, se obtuvo un promedio aproximado de 57 épocas y se estableció un límite máximo de 77 épocas para esta etapa.

Para el entrenamiento final se empleó la totalidad del conjunto de datos disponible, realizando una partición interna del 90 % para entrenamiento y 10 % para validación. Este último subconjunto fue utilizado como referencia para supervisar la evolución del aprendizaje y permitir la actuación del callback EarlyStopping, sin sustituir el procedimiento de validación cruzada realizado previamente.

El entrenamiento finalizó después de 48 épocas debido a la activación del criterio de parada temprana, restaurándose automáticamente los pesos correspondientes a la época 33, en la cual se obtuvo el mejor comportamiento sobre el subconjunto de validación. Finalmente, el modelo resultante fue almacenado en formato .h5 bajo el nombre BuenasMalas_v2_final.h5 para su posterior utilización en las etapas de comparación e integración con el procesamiento de información espectral.

Figura C.5. *Curvas de entrenamiento y validación obtenidas durante el entrenamiento final del modelo RGB.*



Durante el procedimiento de validación cruzada se registró la evolución de las métricas de entrenamiento y validación para cada una de las cinco particiones. Las curvas obtenidas permitieron analizar el comportamiento del proceso de aprendizaje y verificar su estabilidad ante diferentes distribuciones de las muestras, como se presenta en la Figura C.5. Los resultados

cuantitativos derivados de este procedimiento y la evaluación del modelo final se presentan posteriormente en el **¡Error! No se encuentra el origen de la referencia.**

C.2 Adaptación del modelo de detección YOLO al dominio espectral

En este apéndice se describe el proceso mediante el cual el modelo de detección YOLO fue adaptado para operar sobre las representaciones pseudo-RGB construidas a partir de la información espectral adquirida por el sistema. Se presentan el modelo de referencia utilizado como punto de partida, la estrategia de transferencia aplicada para adaptar el modelo al nuevo dominio de entrada, la configuración empleada durante el fine-tuning y, finalmente, el proceso de entrenamiento y almacenamiento del modelo resultante.

C.2.1 Modelo YOLO de referencia

El desarrollo del detector se realizó sobre la arquitectura YOLOv5, implementada mediante el repositorio Ultralytics YOLOv5 v7.0. Se seleccionó la variante YOLOv5m(21.2M parametros) debido al balance que ofrece entre precisión y velocidad de inferencia, característica relevante para un sistema que debe operar en tiempo real sobre una banda transportadora en movimiento continuo.

Como punto de partida no se utilizaron los pesos preentrenados en COCO, sino un modelo YOLOv5m previamente ajustado, mediante fine-tuning, para la detección de tres clases de cítricos (limones, naranjas y mandarinas) sobre imágenes RGB naturales, desarrollado en un trabajo de grado anterior del grupo de investigación HDSP (**¡Error! No se encuentra el origen de la referencia.** Cardozo y Jiménez Buitrago, 2024). Es importante precisar que dicho modelo corresponde a una adaptación de la arquitectura YOLOv5 de Ultralytics mediante transferencia de aprendizaje, y no a una arquitectura de detección desarrollada por el grupo HDSP. Partir de este

modelo permitió aprovechar características visuales generales ya aprendidas para formas y texturas de frutos cítricos, reduciendo el volumen de datos necesario para adaptar el modelo al presente trabajo.

C.2.2 Transferencia del dominio RGB al pseudo-RGB

El modelo de referencia fue entrenado originalmente sobre imágenes RGB convencionales, mientras que el sistema desarrollado en este trabajo requiere detectar mandarinas sobre representaciones pseudo-RGB construidas a partir de tres bandas espectrales seleccionadas (¡Error! No se encuentra el origen de la referencia., sección B.5.3), cuyos canales no corresponden a los valores de color reales de una imagen fotográfica, sino a intensidades de reflectancia en distintas regiones del espectro SWIR.

A pesar de esta diferencia, se consideró que las características visuales de bajo y medio nivel aprendidas por las primeras capas del modelo como bordes, contornos y patrones de forma asociados a frutos redondeados sobre un fondo oscuro seguían siendo relevantes en el nuevo dominio, dado que la geometría y el contraste entre fruto y fondo se conservan en las representaciones pseudo-RGB. Por esta razón, la adaptación del modelo se abordó como un problema de transferencia de aprendizaje, conservando el backbone del modelo de referencia y reentrenando únicamente las capas responsables de la detección, en lugar de entrenar una red completamente nueva desde cero.

C.2.3 Configuración del fine-tuning

La configuración empleada durante el proceso de fine-tuning se resume en la Tabla C.4. El modelo fue ajustado a una única clase (mandarina), congelando las primeras 10 capas

correspondientes al backbone con el propósito de preservar las características generales aprendidas por el modelo de referencia y reentrenar exclusivamente la cabeza de detección.

Tabla C.4 *Parámetros de configuración empleados durante el fine-tuning del modelo YOLOv5m.*

Parámetro	Valor
Framework	Ultralytics YOLOv5 v7.0(commit fe219ed8)
Arquitectura	YOLOv5m
Resolución de entrada	640x640
Batch size	8
Optimizador	SGD (lr=0.01, por defecto)
Programador de tasa de aprendizaje	Cosine annealing (integrado en YOLOv5)
Capas congeladas (--freeze)	10(backbone)
Numero de clases	1 (mandarina)
Épocas máximas	150
Paciencia (early stopping)	25
Anchors	Por defecto de YOLOv5m

C.2.4 Proceso de entrenamiento y almacenamiento del modelo

La cantidad de épocas y el valor del parámetro patience empleados durante el entrenamiento definitivo se establecieron a partir de un entrenamiento preliminar de 80 épocas, en el cual se observó que el modelo continuaba mejorando su desempeño sin que el criterio de early stopping llegara a activarse. Con base en esta observación, se definió un entrenamiento definitivo de 150 épocas utilizando un valor de patience igual a 25, sin realizar una búsqueda sistemática adicional de hiperparámetros.

El entrenamiento se ejecutó en un entorno Google Colab utilizando una GPU NVIDIA Tesla T4 con 14913 MiB de memoria dedicada. La ejecución del entrenamiento definitivo tuvo una duración aproximada de 3,5 horas, durante las cuales el modelo fue ajustado empleando la configuración descrita en la sección anterior. El seguimiento de experimentos mediante la plataforma Weights & Biases (wandb) permaneció deshabilitado durante esta etapa.

Tabla C.5. *Resumen del entrenamiento definitivo del modelo YOLOv5m.*

Parámetro	Valor
Entrenamiento preliminar	80 épocas
Entrenamiento definitivo	150 épocas
Patience	25
Plataforma	Google Colab
GPU	NVIDIA Tesla T4
Tiempo de entrenamiento	3,5 h
Modelo almacenado	best.pt

Al finalizar el entrenamiento, el modelo con mejor desempeño fue almacenado para su posterior utilización durante la etapa de inferencia en tiempo real e integración con el sistema automatizado de clasificación. Las métricas obtenidas durante el entrenamiento, así como las curvas de aprendizaje y la evaluación detallada del detector, se presentan en el [Error! No se encuentra el origen de la referencia..](#)

Para la operación del sistema se definieron los parámetros de inferencia empleados durante la detección automática de las mandarinas. En particular, se establecieron un umbral de confianza (confidence threshold) de 0,60 y un umbral de intersección sobre unión (Intersection over Union,

IoU) de 0,45, configuración que fue utilizada durante las pruebas experimentales y la implementación del sistema sobre la banda transportadora.

C.3 Desarrollo del modelo de clasificación espectral

C.3.1 Estrategia de adaptación al dominio espectral

Una vez obtenido el modelo de clasificación basado en imágenes RGB y validado su funcionamiento, se inició el desarrollo de una estrategia orientada a incorporar la información espectral adquirida mediante la cámara SWIR. El objetivo de esta etapa fue aprovechar la información contenida en las diferentes longitudes de onda para mejorar la capacidad del sistema de discriminar entre mandarinas aptas y no aptas, especialmente en situaciones donde las imágenes RGB presentan limitaciones para representar diferencias relacionadas con el estado superficial del fruto.

A partir del análisis espectral descrito en el [Error! No se encuentra el origen de la referencia.](#) se seleccionaron cinco bandas de interés alrededor de la longitud de onda de máxima respuesta espectral (λ_{pico}). Tres de estas bandas fueron utilizadas para construir una representación pseudo-RGB compatible con modelos de visión por computador preentrenados, mientras que las dos bandas restantes conservaron información espectral adicional destinada a complementar el proceso de clasificación.

Con el propósito de determinar la arquitectura más adecuada para este nuevo dominio de entrada, se implementaron y evaluaron diferentes estrategias de clasificación basadas en

aprendizaje profundo. Todas las arquitecturas fueron entrenadas empleando exactamente el mismo protocolo experimental, de esta manera, las diferencias observadas durante la evaluación pudieron atribuirse principalmente a la arquitectura propuesta y no a variaciones en los datos utilizados para el entrenamiento.

Finalmente, el proceso de desarrollo permitió seleccionar la arquitectura con mayor capacidad de generalización sobre datos no utilizados durante el entrenamiento, la cual fue posteriormente integrada al sistema automatizado de clasificación implementado en este trabajo.

C.3.2 Arquitectura evaluadas

Con el propósito de identificar la arquitectura más adecuada para la clasificación espectral de las mandarinas, se implementaron y evaluaron diferentes estrategias de aprendizaje profundo. Cada una de ellas buscó aprovechar la información espectral mediante enfoques de transferencia de aprendizaje, extracción de características y mecanismos de atención. Para garantizar una comparación objetiva, todas las arquitecturas fueron entrenadas utilizando el mismo protocolo experimental, empleando las mismas particiones de validación cruzada (Stratified K-Fold) y el mismo conjunto Holdout independiente. La Tabla C.6 resume las principales características de las arquitecturas implementadas durante esta etapa del desarrollo.

Tabla C.6. *Arquitecturas evaluadas para la clasificación espectral y principales características.*

Arquitectura	CV (F1)	Holdout (F1)	Selección
Fine-tuning	0.7847	0.7494	—
progresivo			
CNN Dual-Branch	0.8129	0.6000	—

CNN-Transformer	0.8075	0.7749	✓
-----------------	--------	---------------	---

La primera estrategia consiste en un modelo basado en fine-tuning progresivo, en el cual una CNN previamente entrenada sobre imágenes RGB fue adaptada al dominio espectral mediante representaciones pseudo-RGB complementadas con dos bandas espectrales adicionales. El entrenamiento se realizó liberando gradualmente las capas del modelo preentrenado y aplicando técnicas de aumento de datos con ruido gaussiano.

Posteriormente, se implementó una arquitectura Dual-Branch, compuesta por una rama RGB preentrenada y una segunda rama encargada de procesar exclusivamente la información espectral adicional. Ambas ramas fueron fusionadas mediante capas densas para realizar la clasificación binaria, buscando combinar la información espacial y espectral dentro de una misma arquitectura.

Finalmente, se desarrolló una arquitectura híbrida CNN–Transformer, integrada por convoluciones separables, bloques residuales, mecanismos de atención por canales (Channel Attention), atención espacial (Spatial Attention) y un bloque Transformer basado en Multi-Head Self-Attention. Esta arquitectura fue diseñada para mejorar la capacidad de extracción de características y aumentar la robustez del modelo frente a la variabilidad existente entre diferentes sesiones de adquisición.

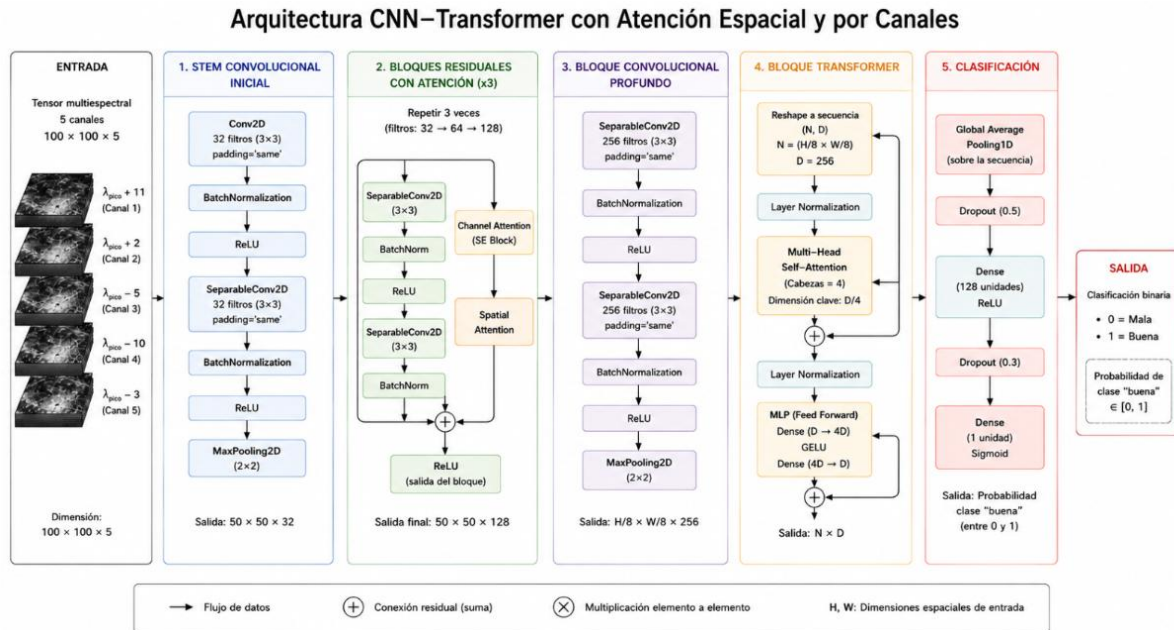
Las diferencias observadas entre las arquitecturas implementadas permitieron analizar distintas estrategias para incorporar información espectral al proceso de clasificación y constituyeron la base para la selección del modelo finalmente integrado al sistema. La descripción detallada de la arquitectura seleccionada se presenta en la siguiente sección, mientras que la

comparación cuantitativa de su desempeño se desarrolla en el ;Error! No se encuentra el origen de la referencia..

C.3.3 Arquitectura del modelo seleccionado

La arquitectura seleccionada corresponde a un modelo híbrido CNN-Transformer diseñado para combinar la capacidad de extracción local de características propia de las redes convolucionales con la habilidad de los mecanismos de atención para modelar relaciones espaciales de mayor alcance. La red recibe como entrada un tensor espectral compuesto por cinco bandas previamente seleccionadas generando como salida la probabilidad de pertenecer a las categorías Apta y no Apta.

Figura C.6. *Arquitectura híbrida CNN–Transformer implementada para la clasificación espectral de mandarinas.*



La red recibe como entrada un tensor de dimensiones $128 \times 128 \times 5$, correspondiente a las cinco bandas espectrales seleccionadas durante el análisis presentado en el ;Error! No se encuentra

el origen de la referencia.. Esta representación conserva la información utilizada posteriormente por los módulos convolucionales y de atención.

La primera parte de la arquitectura está compuesta por un bloque convolucional inicial seguido de tres bloques residuales con convoluciones separables y mecanismos de atención espacial y por canales. Esta etapa permite extraer progresivamente características locales del fruto mientras reduce el costo computacional mediante el empleo de SeparableConv2D.

Las características obtenidas por la etapa convolucional son transformadas en una secuencia que posteriormente es procesada mediante un bloque Transformer basado en Multi-Head Self-Attention. Este módulo permite modelar relaciones entre regiones alejadas de la imagen, complementando la información local aprendida por la CNN.

Finalmente, las representaciones obtenidas son condensadas mediante Global Average Pooling, seguidas por capas densas y regularización mediante Dropout, obteniendo como salida la probabilidad asociada a cada categoría mediante una función de activación Sigmoid.

Tabla C.7. *Componentes principales de la arquitectura propuesta.*

Componente	Función
Stem convolucional	Extracción inicial de características
Bloques residuales	Aprendizaje jerárquico
Channel Attention	Ponderación de canales espectrales
Spatial Attention	Énfasis en regiones relevantes
Transformer	Relaciones globales
Clasificador	Clasificación binaria

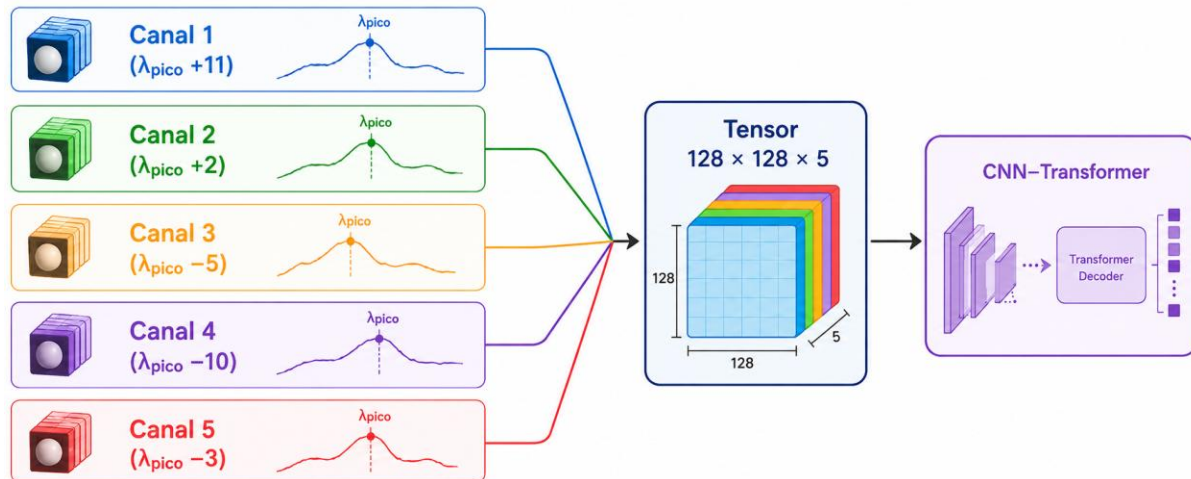
C.3.4 Configuración de la entrada espectral

La arquitectura seleccionada fue diseñada para trabajar directamente con información espectral obtenida a partir de las cinco bandas de interés seleccionadas durante el análisis descrito en el ¡Error! No se encuentra el origen de la referencia.. En lugar de utilizar imágenes RGB convencionales, la red recibe como entrada un tensor multiespectral de dimensiones $128 \times 128 \times 5$, donde cada canal corresponde a una longitud de onda específica alrededor de la respuesta espectral máxima (λ_{pico}).

Las tres primeras bandas ($\lambda_{pico} + 11$, $\lambda_{pico} + 2$, $\lambda_{pico} - 5$) conforman una representación pseudo-RGB que conserva la estructura espacial del fruto, mientras que las dos bandas restantes ($\lambda_{pico} - 10$, $\lambda_{pico} - 3$) aportan información espectral complementaria que no se encuentra presente en una imagen RGB convencional. De esta manera, la red dispone simultáneamente de información espacial y espectral durante el proceso de aprendizaje.

Previo al entrenamiento, todos los tensores fueron normalizados y organizados con un tamaño uniforme de 128×128 píxeles, garantizando que todas las muestras presentaran la misma estructura de entrada. Esta configuración permitió alimentar directamente la arquitectura CNN–Transformer sin requerir etapas adicionales de transformación durante la inferencia.

Figura C.7. *Construcción del tensor multiespectral de entrada para la arquitectura CNN–Transformer.*



C.3.5 Estrategia y etapas de entrenamiento

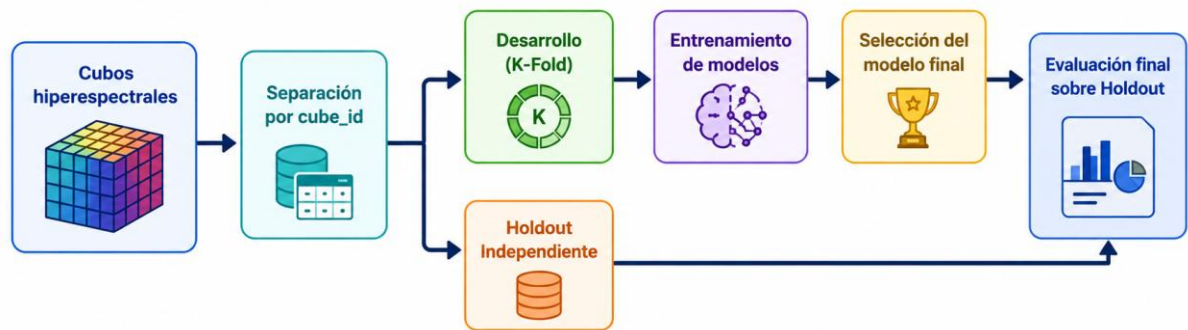
Con el propósito de evaluar de manera objetiva las diferentes arquitecturas propuestas para la clasificación espectral, se estableció un protocolo experimental común para todos los modelos implementados. Cada arquitectura fue entrenada y evaluada utilizando exactamente las mismas particiones de entrenamiento, validación y prueba, garantizando que las diferencias observadas durante la evaluación correspondieran únicamente a la arquitectura empleada y no a variaciones en la distribución de los datos.

Una consideración importante durante esta etapa fue evitar la fuga de información (data leakage) entre los conjuntos de entrenamiento y evaluación. Debido a que una misma mandarina podía aparecer representada en diferentes capturas pertenecientes a un mismo cubo hiperspectral, la partición del conjunto de datos se realizó utilizando el identificador del cubo (cube_id) como unidad de separación. De esta manera, todas las muestras provenientes de un mismo cubo permanecieron siempre dentro del mismo subconjunto de datos.

Posteriormente, sobre el conjunto destinado al desarrollo del modelo se aplicó el mismo protocolo de partición de los datos. Finalizada esta etapa, el modelo seleccionado fue evaluado

utilizando un conjunto Holdout completamente independiente, compuesto por muestras provenientes de sesiones de adquisición diferentes a las empleadas durante el entrenamiento.

Figura C.8. *Protocolo experimental empleado para el entrenamiento y evaluación de las arquitecturas espectrales.*



C.3.6 Configuración del entrenamiento y almacenamiento del modelo

El entrenamiento de la arquitectura CNN–Transformer se realizó completamente desde cero, sin emplear técnicas de transferencia de aprendizaje ni reutilización de pesos previamente entrenados. Durante el entrenamiento se emplearon diferentes estrategias de optimización y control con el propósito de favorecer la convergencia del modelo y reducir el riesgo de sobreajuste.

Tabla C.8. *Configuración del entrenamiento de la arquitectura CNN–Transformer.*

Parámetro	Valor
Framework	TensorFlow / Keras
Arquitectura	CNN–Transformer
Tamaño de entrada	128 × 128 × 5
Optimizador	AdamW

Función de pérdida	Binary Crossentropy
EarlyStopping	Sí
ReduceLROnPlateau	Sí
ModelCheckpoint	Sí
Validación	Stratified K-Fold (5 folds)
Evaluación final	Holdout independiente

Durante el entrenamiento, el callback EarlyStopping permitió detener automáticamente el proceso cuando la pérdida de validación dejó de mejorar, restaurando los pesos correspondientes a la mejor época. Adicionalmente, ReduceLROnPlateau ajustó dinámicamente la tasa de aprendizaje cuando el entrenamiento alcanzó una meseta, mientras que ModelCheckpoint almacenó automáticamente el modelo con mejor desempeño sobre el conjunto de validación para su posterior evaluación sobre el conjunto Holdout.

El modelo finalmente seleccionado fue almacenado para su integración con el sistema automatizado de clasificación. Los resultados experimentales obtenidos durante el entrenamiento y la comparación con las demás arquitecturas evaluadas se presentan en el [Error! No se encuentra el origen de la referencia..](#)