UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS FÍSICAS

DEPARTAMENTO DE
ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

TRABAJO DE FIN DE GRADO

Código de TFG: ACA04

Desarrollo de proyectos de IoT que implementen ML/IA en sistemas empotrados

Development of IoT Projects Implementing ML/AI in Embedded Systems

Supervisor/es: Guillermo Botella Juan

Mario Padilla Pérez

Grado en Física

Curso académico 2024-25

Convocatoria Extraordinaria

Calificación: 9.5

Resumen:

Este trabajo aborda la clasificación de cantos de aves mediante técnicas de aprendizaje automático usando redes neuronales convolucionales (CNN) aplicadas a representaciones espectrales del sonido en forma de espectrogramas de Mel. Debido a la escasez de datos, se recurre al aprendizaje por transferencia (*Transfer-learning*) para el entrenamiento de las arquitecturas. La arquitectura del clasificador se ha determinado mediante técnicas de búsqueda evolutiva; en concreto, a través de un algoritmo genético diseñado para optimizar el rendimiento de la red. Como parte del proceso, se ha construido una base de datos desde cero, implementando técnicas de curado como aumentación y filtrado de datos. Finalmente, se desplegaron tres de los modelos finales en un sistema embebido (NVIDIA Jetson Nano), donde se evaluó su rendimiento como estudio preliminar de la viabilidad previo al despliegue final.

Abstract:

This study addresses the bird sound classification topic using machine learning techniques, specifically convolutional neural networks (CNN) applied to spectral representations of the sound in the form of Mel spectrograms. Due to data scarcity, transfer learning is employed to train the architectures. The classifier architecture was determined using evolutionary search techniques, specifically a genetic algorithm designed to optimize network performance. As part of the process, a database was built from scratch, incorporating data curation techniques such as augmentation and noise filtering. Finally, three of the final models were deployed on an embedded platform (NVIDIA Jetson Nano), where their performance was evaluated as a preliminary assessment of feasibility for the final deployment.

Índice

1. Introducción

El monitoreo de la biodiversidad resulta esencial para el estudio y la conservación de los ecosistemas. Dentro de este campo, la identificación de especies aviares permite evaluar la salud de los entornos naturales, detectar cambios en la fauna local y apoyar proyectos de conservación.

Uno de los métodos más utilizados es el uso de cámaras de fototrampeo, que capturan imágenes activadas por movimiento. Sin embargo, este enfoque presenta ciertas limitaciones: puede depender de condiciones de luz favorables, puede alterar el comportamiento animal, y suele implicar un coste económico elevado si se desea cubrir un área extensa.

Lo que se propone en este texto es una alternativa basada en sensores acústicos. En concreto, se plantea el uso de micrófonos conectados a un sistema embebido en el que se realizará el análisis del paisaje sonoro en tiempo real. En él, se identificarán especies de aves a través de sus cantos mediante técnicas de aprendizaje automático. Este enfoque presenta numerosas ventajas: los micrófonos son omnidireccionales, no están limitados por la luz solar, no interfieren con la fauna y permiten una mayor cobertura territorial a un coste reducido.

El sistema propuesto se compone de los siguientes elementos:

- Micrófonos: i2S MEMS de Adafruit conectados a ESP32.
  Estos serán los encargados de realizar grabaciones continuas del entorno y transmitirlas via Wi-Fi al sistema de procesamiento.
- Unidad de procesamiento: NVIDIA Jetson Nano de 2 GB
  En esta unidad se procesarán y analizarán las grabaciones mediante la ejecución del modelo de clasificación.

Este trabajo se centra exclusivamente en la parte correspondiente al desarrollo del modelo de clasificación y su despliegue en sistemas embebidos. La implementación completa del sistema físico de monitoreo se considera fuera del alcance de esta memoria y se plantea como el paso natural de trabajo futuro.

Por tanto, los objetivos específicos del presente trabajo pueden resumirse en los siguientes:

- Creación y curado de una base de datos propia.
- Evaluación de arquitecturas convolucionales mediante algoritmos evolutivos
- Despliegue y optimización de diferentes modelos finales en la unidad de procesamiento embebida.

Para analizar las grabaciones, se ha optado por utilizar espectrogramas de Mel, una representación ampliamente empleada en el estado del arte por su capacidad de capturar las características relevantes del sonido, sobre todo, en entornos bioacústicos.

Relativo a la elección de los modelos de red neuronal elegidos para clasificación, estos se han seleccionado en función de las estructuras que mejores resultados han proporcionado para el *dataset* creado, compuesto de 20 especies: 12 diurnas y 8 nocturnas. A la hora de construir el *dataset* se han tenido en cuenta los problemas comunes de cualquier investigador: desbalance de clases y un conjunto pequeño de datos.

Para seleccionar la arquitectura óptima se ha hecho uso de un algoritmo genético buscando la mejor arquitectura con *transfer-learning*, compuesta por un *backbone* preentrenado y capas adicionales, más adelante se detallará el espacio de búsqueda.

Por último, se ha realizado un estudio para implementar los mejores modelos en nuestro sistema embebido. En concreto, se han desplegado arquitecturas de diferentes tamaños con el objetivo de tener una estimación del rendimiento para diferente complejidad computacional.

En la siguiente sección se detallan los conceptos teóricos que deben conocerse para entender el desarrollo del resto de la memoria.

2. Marco teórico

2.1. Procesamiento de audio para reconocimiento de sonidos

El análisis automático de señales acústicas es fundamental en diversas tareas de clasificación y detección de eventos sonoros, como es el caso que tratamos en este trabajo: el reconocimiento de aves por su canto.

Las señales de audio crudas (amplitud vs. muestras) contienen la totalidad de la información acústica; sin embargo, su alta dimensionalidad y la falta de estructura espacial, complican el procesamiento directo de las mismas mediante redes neuronales. Por ello, es común transformar estas señales a representaciones en el dominio tiempo-frecuencia que permitan extraer características relevantes de manera más eficiente.

Entre estas representaciones, una de las más utilizadas es el espectrograma de Mel, una variante del espectrograma de corta duración (STFT) que incorpora una escala perceptual inspirada en la audición humana. El proceso de obtención del espectrograma de Mel se realiza en dos etapas: primero, se aplica la transformada rápida de Fourier (FFT) en ventanas temporales sucesivas para obtener un espectrograma clásico (STFT); después, se aplica un banco de filtros triangulares distribuidos logarítmicamente según la escala de Mel. El resultado es una matriz bidimensional que representa la energía espectral de la señal a lo largo del tiempo en distintas bandas de frecuencia perceptuales.

Otra representación ampliamente explorada en la literatura, véase [17] por ejemplo, son los espectrogramas STFT, ya mencionados. El resultado es una matriz en la que cada columna representa el espectro de frecuencia de una ventana temporal de la señal. En este caso, la distribución de frecuencias es lineal.

También se tratan ampliamente los MFCC (Coeficientes Cepstrales en las Frecuencias de Mel, por sus siglas en inglés) en contextos bioacústicos, como tareas de reconocimiento del habla. Estos son coeficientes derivados del espectrograma de Mel a través de una Transformación de Coseno Discreta (DCT) y almacenan de manera aún más compacta la información espectral. Sin embargo, estos no preservan la información espacial de la señal, lo que puede limitar su utilidad en contextos en los que los patrones espectro-temporales sean fundamentales.

La elección del uso de espectrogramas de Mel para tareas de clasificación de cantos de aves, en este trabajo, se fundamenta en su capacidad para capturar de forma eficiente la estructura espectro-temporal del sonido. Para ello, utilizan una escala perceptual que emula la percepción auditiva de muchos vertebrados. Además, estos espectrogramas permiten resaltar patrones acústicos relevantes para la identificación de especies, reducen la dimensionalidad del espectrograma original y son robustos frente al ruido.

Por otra parte, las redes neuronales convolucionales (CNN) han demostrado ser especialmente eficaces para procesar entradas en forma de imágenes bidimensionales, como lo son los espectro-

gramas. Las CNN son capaces de aprender filtros locales que capturan patrones discriminativos de frecuencia y tiempo, por ejemplo estructuras rítmicas, sin requerir procesamiento manual. Además, al representar el audio en forma de imagen, se pueden usar arquitecturas convolucionales preentrenadas en conjuntos de datos visuales, como ImageNet, [3], mediante técnicas de aprendizaje por transferencia. Esto permite obtener buenos rendimientos con cantidades limitadas de datos acústicos. Por todo esto, las CNN se han consolidado como una de las arquitecturas más utilizadas en la literatura para tareas de clasificación de sonidos y bioacústica.

2.2. Aprendizaje por transferencia

El aprendizaje por transferencia o *transfer learning* es una técnica que permite reutilizar el conocimiento adquirido por un modelo previamente entrenado en una tarea fuente sobre otra tarea destino relacionada. Esta estrategia es especialmente relevante en contextos en los que prima la escasez de datos etiquetados.

En este trabajo, se emplean arquitecturas convolucionales preentrenadas en ImageNet, cuyo módulo convolucional se mantiene como extractor de características para un clasificador completamente conectado que situaremos al final de la arquitectura convolucional importada; un ejemplo de un caso simple puede verse en [2] [1].

Más específicamente, sobre la salida del cuerpo convolucional, se incorpora una capa de promediado (*average pooling*) bidimensional con el objetivo de reducir la dimensionalidad y conservar la información más relevante. A continuación, se incluyen capas completamente conectadas con o sin regularización, mediante *dropout* y, finalmente, una capa de salida. De esta manera, el bloque convolucional extrae los rasgos más relevantes del audio, mientras que las capas superiores adaptan la red a la tarea específica que se trata en este trabajo, la clasificación de aves mediante sus cantos.

En resumen, las principales ventajas de la reutilización de modelos preentrenados son las siguientes: una reducción de la cantidad necesaria de datos para alcanzar resultados competitivos, una aceleración de la convergencia de los modelos y una mejora de la generalización en contextos con datos limitados.

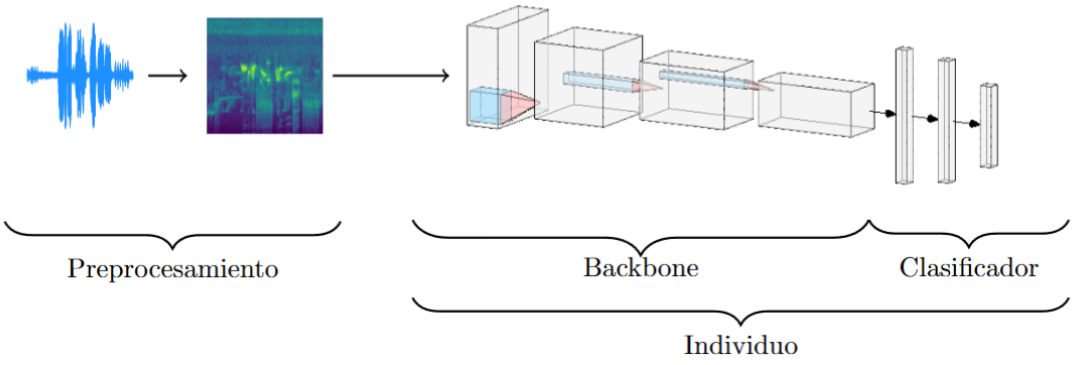

Figura 1: Diagrama de flujo del sistema

[1] Cabe mencionar que en el presente trabajo no se considera el ajuste fino o *fine-tuning* tras el entrenamiento del clasificador; dejándose como trabajo futuro para mejorar aún más los resultados obtenidos

2.3. Algoritmos para la búsqueda de arquitecturas (NAS)

En primer lugar, *Neural Architecture Search* (NAS) es una técnica de *AutoML* [2] que automatiza el diseño de redes neuronales para una tarea determinada. En los últimos años, su popularidad ha crecido de forma significativa, impulsada por su capacidad para superar el rendimiento de arquitecturas diseñadas manualmente en numerosos campos. Esto ha consolidado a NAS como un enfoque emergente y cada vez más dominante en la construcción de modelos de aprendizaje automático [15].

Los métodos de la búsqueda automática de arquitecturas se pueden englobar por las siguientes características:

- **Naturaleza del espacio de búsqueda:** atañe a las configuraciones de arquitecturas, que pueden estructurarse a nivel de bloques, *micro search space*, o de redes completas, *macro search space*.
- **Estrategia de búsqueda:** método utilizado para explorar el espacio, incluyendo algoritmos evolutivos, aprendizaje por refuerzo y optimización bayesiana, entre otros.
- **Estimación de rendimiento:** técnicas que permiten evaluar la calidad de una arquitectura sin necesidad de entrenarla completamente. Entre estas se incluyen entrenamiento parcial, predictores de rendimiento o técnicas *one-shot*, entre otros.

En este trabajo se ha elegido un algoritmo evolutivo; en particular, un algoritmo genético, cuya descripción se detalla en 3.2. Con él, se busca hallar el mejor clasificador para cada elemento de una serie de estructuras convolucionales preentrenadas. El rendimiento se estima mediante un breve y reducido entrenamiento del clasificador acoplado a la estructura base, en adelante referida como *backbone*.

Los algoritmos genéticos son una alternativa robusta y competitiva en NAS, como señalan White et al. (2023) [15]; además, la facilidad de implementación, su gran flexibilidad a la hora de incorporar múltiples objetivos y restricciones, y su simplicidad conceptual han reforzado esta elección.

2.4. Despliegue en sistemas embebidos: consideraciones y métricas

El despliegue de CNN en sistemas empotrados plantea los problemas típicos: capacidad de cómputo limitada, escasa memoria disponible y bajo presupuesto energético. Por ello, priman los modelos ligeros, eficientes y optimizados para lograr un desempeño aceptable sin comprometer la viabilidad del sistema.

Por ende, las métricas principales que vamos a monitorizar son:

- **Latencia:** tiempo promedio por inferencia.
- **Throughput:** número de inferencias por segundo.
- **Uso de memoria:** memoria RAM utilizada durante la inferencia.
- **Uso de CPU y GPU:** medido para evaluar la eficiencia de cómputo.

Además, estas métricas están estrechamente ligadas con las métricas del modelo y viceversa:

- **Métrica de rendimiento:** en nuestro caso elegimos *F1-score* por ser la media aritmética entre precisión y sensibilidad.

[2] Así es como se denomina al proceso de automatización de los pasos de una metodología de *Machine Learning*, como el curado de datos, selección de hiperparámetros, etc.

- **Parámetros del modelo:** permiten estimar la memoria no volátil requerida para su almacenamiento.
- **FLOPs:** número de operaciones aritméticas que precisa la red para procesar una entrada, lo que nos da una aproximación del coste computacional del modelo.
- **Representación de datos numéricos:** tipo de codificación empleado para representar pesos y activaciones del modelo (`int8`, `fp16` o `fp32`, entre otros). Estos impactan directamente en el resto de parámetros del modelo que, a su vez, influyen en los parámetros de despliegue.

En particular, se estudiará el despliegue de la selección de algunas de las mejores arquitecturas encontradas mediante un algoritmo genético en una NVIDIA Jetson Nano de 2 GB de memoria RAM. En este entorno, se analizarán las métricas mencionadas con el objetivo de evaluar las arquitecturas encontradas bajo restricciones reales.

3. Metodología

3.1. Base de datos

3.1.1. Selección y obtención de grabaciones

Para desarrollar un sistema capaz de monitorizar el paisaje sonoro de forma continua, es fundamental construir un *dataset* representativo tanto de la actividad diurna como nocturna. Se seleccionaron cuidadosamente un total de 20 especies de aves: 12 diurnas y 8 nocturnas. Además, se añadió una clase adicional correspondiente a eventos no aviares, que se alinea con el propósito de analizar grabaciones de manera ininterrumpida.

Entre las especies diurnas incluidas se encuentran: *Upupa epops* (Abubilla), *Certhia brachydactyla* (Agateador común), *Parus major* (Carbonero común), *Myiopsitta monachus* (Cotorra argentina), *Passer domesticus* (Gorrión común), *Carduelis carduelis* (Jilguero), *Turdus merula* (Mirlo común), *Columba palumbus* (Paloma torcaz), *Erithacus rubecula* (Petirrojo europeo), *Picus viridis* (Pito real), *Pica pica* (Urraca común) y *Cyanistes caeruleus* (Herrerillo común).

Por otro lado, las especies nocturnas seleccionadas fueron: *Bubo bubo* (Búho real), *Asio otus* (Búho chico), *Strix aluco* (Cárabo común), *Caprimulgus europaeus* (Chotacabras europeo), *Tyto alba* (Lechuza común), *Nycticorax nycticorax* (Martinete común), *Athene noctua* (Mochuelo europeo) y *Otus scops* (Autillo europeo).

Las grabaciones de aves se obtuvieron íntegramente de xeno-canto [16], seleccionando aquellas de mayor calidad [3]. Complementariamente, para la clase adicional, se seleccionaron manualmente las grabaciones para incluir aquellos eventos acústicos más relevantes de los posibles entornos en los que podría desplegarse el sistema. Esto es una práctica bastante común, véase por ejemplo [5].

3.1.2. Procesamiento

- **Etiquetado:** pese a que en el estado del arte, véase BirdCLEF [6], se trate el problema completo, mediante aprendizaje débilmente supervisado, en este trabajo se ha optado por un enfoque más sencillo. Se considera una única etiqueta por muestra, conocida con certeza, lo que permite formular el problema como una clasificación multiclase supervisada.

[3] Este parámetro puede tomar los valores entre A (máxima calidad) y E (mínima calidad). Aquí se encapsula información acerca de la nitidez de la grabación, el ruido de fondo y la identificabilidad de la llamada

- **División del conjunto de datos:** se realiza una división estratificada por especie, asegurando que cada subconjunto en el que se divide la base de datos herede las proporciones de la base de datos original. Estos subconjuntos son: entrenamiento (70 %), validación (15 %) y prueba (15 %).

- **Construcción de espectrogramas:**

  - Duración de los segmentos: 5 segundos, un valor razonable para incluir fragmentos identificables de patrones de canto y comúnmente usada en el estado del arte.
  - Tasa de muestreo: sr = 32 kHz.
  - Número de bandas de mel: n_mel = 128, lo que proporciona un buen balance entre resolución perceptual y complejidad computacional.
  - Tamaño de la ventana *FFT*: n_fft = 1024, esto equivale a una ventana temporal de unos 32 ms para *sr*, suficiente para distinguir estructuras relevantes.
  - Desplazamiento entre ventanas: hop_length = 500, resultando en una resolución temporal de unos 16 ms con un 50 % de superposición.
  - Rango de frecuencias: f_min = 40 Hz y f_max = 15 kHz.

Figura 2: Forma de onda

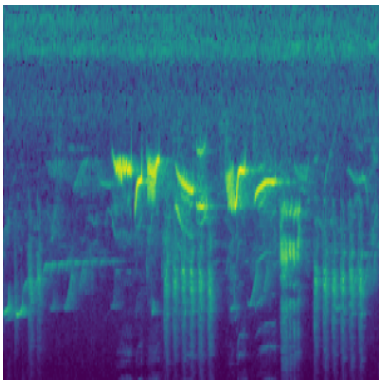

Figura 3: Espectrograma de Mel.

- **Filtrado**: para mejorar la calidad del conjunto de datos y reducir la presencia de muestras no relevantes, se aplicó un filtro preliminar utilizando una CNN binaria entrenada para distinguir entre dos clases: *pájaro* y *ruido*. Esta red permitió eliminar aquellos espectrogramas dominados por ruido ambiental o vocalizaciones poco claras; es decir, con baja relación señal-ruido.

  Para su entrenamiento, se empleó una porción representativa de espectrogramas de cada especie de ave del *dataset* y elementos de la clase *fondo*, a los que se añadieron espectrogramas mal etiquetados. Se usó EfficientNetB0 como *backbone* por su ligero coste computacional y su buen rendimiento en problemas de clasificación visual.

  Además de enfoques basados en NNs discriminativas, como el empleado, en la literatura se abordan diversos métodos para el preprocesamiento y filtrado de datos bioacústicos. Entre ellos se encuentran los métodos basados en energía o detección de actividad (VAD), que emplean métricas como la energía por frame o la entropía espectral para detectar eventos acústicos relevantes [17]. También son comunes técnicas de reducción de ruido espectral, como

el filtro Wiener, que se aplica a la señal original para mejorar la relación señal-ruido. Más recientemente, se están explorando métodos prometedores basados en separación de fuentes.

- **Aumentación de datos:** el uso de técnicas de aumentación de datos ha sido fundamental en el desarrollo de los modelos de clasificación de sonidos ambientales y biológicos.

  Salamon y Bello (2017), [13], fueron pioneros en implementar estas técnicas, demostrando su importancia para mejorar el rendimiento de los modelos.

  Más tarde, Nanni et al. (2019), [11], evidenciaron que las aumentaciones convencionales de imágenes, e.g. reflexiones, ofrecían peores resultados que directamente no haber realizado ninguna aumentación. Además, mediante el ensamblado de modelos a los que se les había practicado una sola familia de aumentaciones, consiguieron mejorar los resultados del estado del arte sin optimizar los parámetros de las redes neuronales.

  Continuando con estos resultados, recientemente, Kumar et al. (2024), [8], propusieron una *pipeline* de aumentaciones múltiples que presentaba mejores resultados que aumentaciones simples; es decir, una por espectrograma. Concluyendo además que las mejores aumentaciones simples son aquellas que introducen ruido: distorsión de tanh, adición de ruido al uso, cambios en la ganancia, etc.

  El enfoque elegido sigue estas pautas; en concreto, se implementa algo similar a la propuesta de Kumar et al. considerando aumentaciones múltiples y dando gran importancia a las transformaciones que incluyen alguna clase de ruido. Además, también se incluyen transformaciones frecuentemente empleadas en BirdCLEF. En concreto:

  - Suma de audios de la misma clase [4]
  - Ruido de fondo
  - Cambios en ganancia
  - Cambios en el tono
  - Cambios de velocidad
  - Distorsión de Tanh
  - Normalización
  - Padding (relleno) solo para $t < 5$ s

3.2. Selección de CNN mediante búsqueda evolutiva

3.2.1. Objetivo

En esta fase del trabajo se busca seleccionar las arquitecturas más prometedoras para la tarea de clasificación propuesta.

Para ello, primero se realizó un estudio preliminar de diferentes *backbones*, véase Tabla 1, donde se eligieron los más afines con el enfoque del trabajo. Seguidamente, se definió un espacio de búsqueda de hiperparámetros, adecuado para el problema, y se ejecutó un algoritmo genético para encontrar las arquitecturas más relevantes.

3.2.2. Detalles del algoritmo evolutivo

El algoritmo evolutivo que se ha implementado es muy similar al propuesto por Real et al. (2019), [12], que, como se menciona en [15], consiguió resultados del estado del arte sobre ImageNet en el momento de su publicación y continúa siendo uno de los algoritmos genéticos más eficaces a día de hoy a la hora de encontrar arquitecturas prometedoras.

Este algoritmo genético utiliza mutación como mecanismo evolutivo. Mantiene el tamaño de la población constante, P, creando un solo individuo por generación y aplicando envejecimiento FIFO ("Primero en entrar, primero en salir", por sus siglas en inglés). La selección de los padres se realiza por torneos de tamaño S, véase Algoritmo 1.

[4] A excepción de esta transformación, el resto se encuentran recogidas en Audiomentations [1].

Algoritmo 1 Regularized Evolution (Real et al., 2019)

Input: Tamaño de población P, tamaño del torneo S
Inicializar una población vacía $\mathcal{P}$
for $i = 1$ to P **do**
    Crear un modelo aleatorio m
    Entrenar m y evaluar su exactitud
    Añadir m a $\mathcal{P}$
end for
while criterio de parada no se cumpla **do**
    Seleccionar aleatoriamente S individuos de $\mathcal{P}$
    Elegir el mejor modelo m_{best} del torneo
    Clonar y mutar m_{best} para crear m'
    Entrenar y evaluar m'
    Añadir m' a $\mathcal{P}$
    Eliminar el individuo más antiguo de $\mathcal{P}$
end while
Return: Mejor modelo encontrado

En el contexto de *transfer learning*, el espacio de soluciones presenta una fuerte discretización, ya que cada *backbone* preentrenado define una subregión del espacio en la que el balance entre exactitud, tamaño y simplicidad computacional es óptimo. Para mitigar el sesgo hacia *backbones* dominantes, se optó por desactivarlos en etapas avanzadas de la evolución, promoviendo así la exploración de individuos más débiles [5].

3.2.3. Descripción del procedimiento seguido

- **Hardware, framework y política de entrenamiento:** la totalidad del entrenamiento ha sido realizado en kaggle. Se usaron dos GPUs NVIDIA T4 de 16 GB de RAM asociadas en paralelo. El *framework* de trabajo ha sido TensorFlow Keras. Además, respecto a la política de entrenamiento, se usó precisión mixta (`mixed_fp16`), ver Micikevicius et al. (2018) [10], para acelerar el entrenamiento y reducir el tamaño de los modelos finales.

- **Preselección de *backbones*:** se realizó un estudio preliminar en un amplio rango de parámetros y FLOPs de diferentes estructuras preentrenadas en ImageNet. Estas fueron entrenadas a 5 épocas con el *dataset* original para filtrar las de mejor compromiso entre exactitud, tiempo por época, complejidad computacional y tamaño. En la selección también intervinieron las limitaciones del hardware con el que se trabaja.

  Los resultados de estas evaluaciones preliminares, se encuentran expuestos en la Tabla 1. En ella se muestran la exactitud y la complejidad computacional, y se resaltan las arquitecturas que han sido elegidas.

- **Definición del espacio de búsqueda por *backbone*:** una vez encontrados los *backbones* con los que se va a trabajar [6], hay que determinar la estructura adicional que los va a acompañar. Para ello, a partir de una exploración preliminar, se presentan las siguientes restricciones:

[5] La alternativa metodológicamente más rigurosa es realizar una búsqueda evolutiva completamente diferente para cada *backbone*, lo cual podría descubrir combinaciones aún más especializadas. No se llevó a cabo por restricciones de tiempo de cálculo.

[6] Se encuentran señalados en Tabla 1.

Tabla 1: Comparación de *backbones* en términos de exactitud, número de parámetros, FLOPs y tiempo por época de entrenamiento.

Backbone	**Exactitud (%)**	**Parámetros**	**FLOPs**	**Tiempo/época (s)**
MobileNetV3Small	**76.58**	**0.95M**	**0.12G**	**88.32**
MobileNetV2	46.69	2.28M	0.61G	94.29
MobileNetV3Large	**83.12**	**3.02M**	**0.44G**	**105.93**
EfficientNetB0	**79.42**	**4.08M**	**0.80G**	**94.78**
NASNetMobile	34.14	4.29M	1.15G	107.47
EfficientNetV2B0	**80.77**	**5.95M**	**1.46G**	**108.43**
EfficientNetV2B1	**79.95**	**6.96M**	**2.42G**	**115.06**
DenseNet121	63.40	7.06M	5.70G	142.07
EfficientNetV2B2	**80.48**	**8.80M**	**3.44G**	**123.70**
EfficientNetB3	**81.30**	**10.82M**	**3.76G**	**148.06**
EfficientNetV2B3	**81.21**	**12.96M**	**6.09G**	**121.19**
EfficientNetB4	79.47	17.71M	9.02G	260.08
EfficientNetV2S	78.26	20.36M	16.89G	222.00
InceptionV3	59.82	21.85M	11.47G	136.50
ResNet50V2	56.35	23.61M	6.99G	105.38
ConvNeXtTiny	**78.39**	**27.84M**	**8.78G**	**98.53**
EfficientNetB5	80.96	28.56M	21.00G	496.47

  - Los individuos sin capas densas adicionales presentan, en general, peor rendimiento que los demás. Por simplicidad, se elige además un número de capas densas inferior a 3, excluyendo la capa de salida.
  - El número de neuronas por capa se restringió al conjunto $\{64, 128, 256, 512\}$.
  - Las tasas de dropout elegidas son $\{0{,}0, 0{,}2, 0{,}3, 0{,}4\}$.
  - Se descartaron estructuras consideradas no convencionales, orden creciente en neuronas (`Dense(128) → Dense(256)`) o en dropout.

  Con estas restricciones, el número de configuraciones posibles quedó reducido a 1024.

- **Ejecución del algoritmo genético:** para agilizar todo lo posible la evaluación, se usó un subconjunto balanceado del 10 % del *dataset* original y se entrenó a 5 épocas. Por otro lado, se fijó el tamaño de la población inicial a 50 individuos y el tamaño de torneo a un valor de 7. El algoritmo evolutivo se ejecutó un total de 222 generaciones, hasta que todos los mejores individuos fueron eliminados por FIFO. En total, se evaluaron 272 individuos, aproximadamente un 26 % del espacio de búsqueda. La ejecución tuvo una duración total de unas 9 horas de cálculo.

  Es importante señalar que el uso de un subconjunto reducido y un entrenamiento tan breve introduce sesgos en la estimación del rendimiento de los individuos; en particular, penaliza a las estructuras más lentas en converger. No obstante, estas condiciones se asumieron razonables para la fase inicial de exploración.

- **Selección de la mejor arquitectura por *backbone*:** tras ejecutar el algoritmo genético, se seleccionaron las 3 mejores arquitecturas por *backbone* y se entrenaron en una fracción del 50 % del conjunto de entrenamiento por un total de 15 épocas donde, ahora sí, se consideró el desbalance de clases. En total, se evaluaron 27 arquitecturas.

Los criterios que se usaron en esta segunda selección fueron variados. Se tuvieron en cuenta, de nuevo, el tamaño, el número de FLOPs y el tiempo por época. Además, se consideró también el potencial de aprendizaje a futuro, mediante las curvas de *pérdida* en entrenamiento y validación, y la evolución de *F1-score* en validación, así como su valor máximo.

- **Entrenamiento de los mejores individuos:** por último, los mejores individuos fueron entrenados hasta un máximo de 100 épocas en el *dataset* completo. Para mitigar el desbalance de clases se usa *focal loss* y se asignan pesos a cada una de las clases. Respecto al resto del entrenamiento, se usó *early stopping* con una paciencia de 7 épocas y se usó un planificador de la tasa de aprendizaje [7], *ReduceLROnPlateau* con una paciencia de 3 épocas. La elección de este planificador respecto a otros, como *CosineAnnealing*, se debió simplemente a su flexibilidad para las diferentes estructuras consideradas y a la facilidad de implementación en el entorno de desarrollo.

3.3. Despliegue de los modelos en un sistema embebido

Tras el entrenamiento completo de los modelos, se seleccionarán varios de los modelos finales entrenados y se desplegarán en la placa de desarrollo que estudiamos: NVIDIA Jetson Nano (2 GB). Se hará una comparación de 2 versiones del modelo [8]: modelo sin optimizar en `fp32` y modelo optimizado con precisión mixta en `fp16` y `fp32` y se expondrán las métricas mencionadas en 2.4. Además, se comparará el desempeño de los modelos bajo los dos modos de potencia con los que puede configurarse el rendimiento de la placa y se realizará un estudio en diferentes condiciones térmicas para simular el entorno de despliegue.

Debido a que no hay una *pipeline* completa de detección de cantos de aves, las métricas extraídas en esta sección están referidas únicamente al estudio del modelo, no consideran preprocesamiento de datos ni comunicación con otros sistemas; solo inferencia.

4. Resultados

4.1. Algoritmo genético

4.1.1. Resultados de la ejecución

Como ya se ha mencionado en la sección anterior, la población inicial se eligió de tamaño 50 y el algoritmo se ejecutó hasta que todos los mejores individuos fueron eliminados de la población, lo que tomó 222 generaciones. En total, se evaluaron 272 individuos y el proceso tomó alrededor de 9 horas.

Los resultados de la ejecución están expuestos en Fig. 4.

4.1.2. Filtrado posterior

En esta etapa se seleccionaron los 3 mejores individuos por arquitectura basándonos únicamente en la exactitud sobre el conjunto de validación. Estos resultados están expuestos en la Tabla 2.

[7] Esta herramienta es la encargada de variar la tasa de aprendizaje o *learning rate* según un criterio. El elegido, *ReduceLROnPlateau* reduce a la mitad la tasa de aprendizaje cuando el parámetro que se monitoriza no mejora en un número dado de épocas, la paciencia.

[8] Cabe mencionar que, pese a que `TensorRT` soporte precisión mixta, ha habido problemas derivados de la imposición de este tipo de *política* en el entrenamiento a la hora de ser optimizados dando lugar a modelos más rápidos, pero con una pérdida total de exactitud.

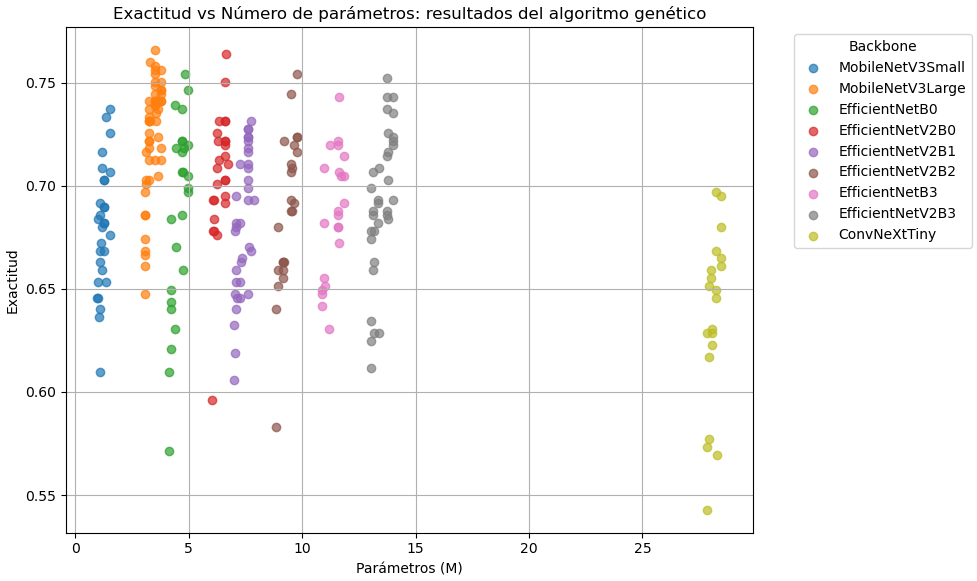

Figura 4: Individuos evaluados por el algoritmo genético

Como puede observarse en la Tabla 2, el sesgo introducido en el proceso de búsqueda ha favorecido la generación de individuos con una gran cantidad de neuronas y tasas de dropout bajas, es decir, arquitecturas que aprenden mucho y muy rápido.

A partir de estos resultados, pueden inferirse varias conclusiones. En primer lugar, la competición *head-to-head* entre diferentes *backbones* no ha sido justa. Una comparación entre individuos con la misma arquitectura base podría haber proporcionado resultados más consistentes y representativos. Además, sumado a esto, podrían especializarse más los parámetros de búsqueda para cada uno, considerando quizás la implementación de una búsqueda multiobjetivo personalizable a cada caso.

En segundo lugar, se observa una clara dominancia de las arquitecturas más rápidas. Un enfoque alternativo basado en un entrenamiento a mayor número de épocas, combinado con una estrategia de *Early Stopping* [9], podría haber resultado más beneficioso que entrenamientos rápidos y filtrado posterior.

Para seleccionar la arquitectura final tras 15 épocas de entrenamiento, se consideraron principalmente tres criterios: desempeño (F1-score), eficiencia computacional (número de parámetros, FLOPs y tiempo por época) y tendencia al sobreajuste (observada en curvas de pérdida y regularización aplicada).

En las variantes de MobileNetV3, se priorizó elegir configuraciones que mantuvieran un buen compromiso entre rapidez y ligereza, dada la naturaleza de estos backbones diseñados para dispositivos con recursos limitados.

Para EfficientNetB0, dado que pertenece también a la categoría de backbones ligeros pero con mayor capacidad, se seleccionó la arquitectura más pesada y con mejores resultados. La variante más ligera muestra una regularización excesiva y una convergencia más lenta, lo que podía limitar su rendimiento final. Respecto a la estructura mediana, el coste añadido de elegir la estructura más pesada es marginal frente a la mejora en desempeño.

[9] *Early Stopping* es un método de regularización cuyo propósito es evitar el sobreajuste. Esto se hace considerando la mejora de un parámetro ajeno al conjunto de entrenamiento. La paciencia en este método atañe al número de épocas en las que se espera para una mejora, si se sobrepasa, se para el entrenamiento.

En EfficientNetV2B0 y V2B1, se descartaron configuraciones sin regularización que mostraban inicio temprano de sobreajuste, optando por individuos con un buen equilibrio entre capacidad y regularización, además de priorizar aquellos con menor tiempo por época.

Para el resto de backbones pesados: EfficientNetV2B2, EfficientNetB3, EfficientNetV2B3 y ConvNeXtTiny, se favorecieron configuraciones con mayor regularización y mayor número de parámetros, dado que estas variantes más pesadas podían beneficiarse de esta regularización para evitar el sobreajuste y mejorar la generalización sin comprometer demasiado el tiempo de entrenamiento.

4.1.3. Entrenamiento de los mejores individuos hallados

Como se mencionó en 3.2, el entrenamiento se hace a 100 épocas con *Early Stopping* y con *ReduceLROnPlateau*, como planificador de la tasa de aprendizaje . Como optimizador del clasificador se usó *AdamW*, una variación de *Adam* que incluye regularización. Además, se usó precisión mixta en el entrenamiento. Como tamaño de lote o *batch size* se usó 128.

Los únicos parámetros que no fueron generales para todos los modelos fueron las tasas de aprendizaje y las tasas decaimiento de pesos según el tamaño del modelo.

Los resultados y las métricas del entrenamiento están expuestos en la Tabla 3, y los resultados de la evaluación sobre el conjunto de prueba, en la Tabla 4.

Se observa en la Tabla 3 que, en general, los modelos más ligeros están sobreajustados como resultado de los sesgos que se introdujeron anteriormente. No obstante, se han encontrado arquitecturas que no sobreajustan en exceso y dan muy buenos resultados sin necesidad de cambiar los hiperparámetros, como es el caso de EfficientNetV2B0 y EfficientNetV2B1. Hay un caso donde hay un subajuste más marcado, EfficientNetV2B2, debido en parte a la gran tasa de *dropout* que presenta.

El resto de individuos deberían poder beneficiarse de una mayor regularización, ya sea aumentando datos, variando la tasa de *dropout* o variando la tasa de decaimiento de pesos; o mediante una disminución del número de parámetros entrenables, que se consigue reduciendo el tamaño del clasificador.

Cabe mencionar que el modelo con peores resultados es, curiosamente, el que más parámetros y mayor capacidad de aprendizaje presenta. Esto es consecuencia de la simplicidad del conjunto de datos y su breve extensión, para este modelo. Además, en lugar de haber seleccionado un individuo con menor número de parámetros entrenable en la fase previa, se seleccionó al más pesado porque presentaba *dropout* pero, la opción más ligera podría haber sido una buena elección en este caso.

Estos resultados dejan entrever que los *backbones* más ligeros se beneficiarán de clasificadores más grandes y de tasas de *dropout* más altas que los *backbones* más pesados, que pueden beneficiarse en mayor medida de un buen ajuste fino que de un clasificador. Aunque esto no quiere decir que los *backbones* sin clasificador sean mejores en ningún caso que las arquitecturas propuestas; de hecho, se entrenó bajo las mismas condiciones a MobileNetV3Small, EfficientNetV2B0 y ConvNeXtTiny y se observaron resultados peores, del orden de un 10 % en *F1-score* en todos los casos.

Respecto a los resultados en el conjunto de prueba, expuestos en la Tabla 4, estos son buenos en general: resultados de exactitud, precisión y sensibilidad macro están en torno a 90 % en todos los modelos, salvando el caso de EfficientNetB*. Asimismo, top 3 score es superior al 95 % en, prácticamente, todos los casos. No obstante, hay una clase en particular que presenta los peores valores de recall en la mayoría de modelos, *Picus viridis*. Esto es curioso, porque la precisión es alta. Viendo los espectrogramas, observamos que un número considerable de ellos no presenta cantos de este pájaro, sino canto de otras aves. Por eso, la cantidad de falsos positivos es baja y la cantidad de falsos negativos es alta, porque hay espectrogramas etiquetados como *Picus Vi-*

ridis que no se corresponden a esta clase. Consecuencia de un filtrado ineficiente para este escenario.

Son curiosos los resultados pobres de la arquitectura EfficientNetB*, en el conjunto de prueba, respecto a lo que se esperaba teniendo en cuenta las métricas de validación. Además, *Myiopsitta Monachus* es la clase menos representada y ambas estructuras han tenido más problemas a la hora de clasificarla, en comparación con otras especies.

Finalmente, debe hacerse un comentario sobre el conjunto de datos y la calidad del filtrado, ya que ahora estamos en posición de poder evaluarlo. Observando Fig. 5, vemos que se clasifican erróneamente aves que conviven en el mismo entorno. Esto se debe, mayoritariamente, a que en el curado del conjunto de datos no se consideró relevante este fenómeno. Sin embargo, tras ver los resultados y el impedimento para seguir mejorando más allá del 90 % generalizado en todos los modelos, es inherente la necesidad de una mejor limpieza del *dataset*.

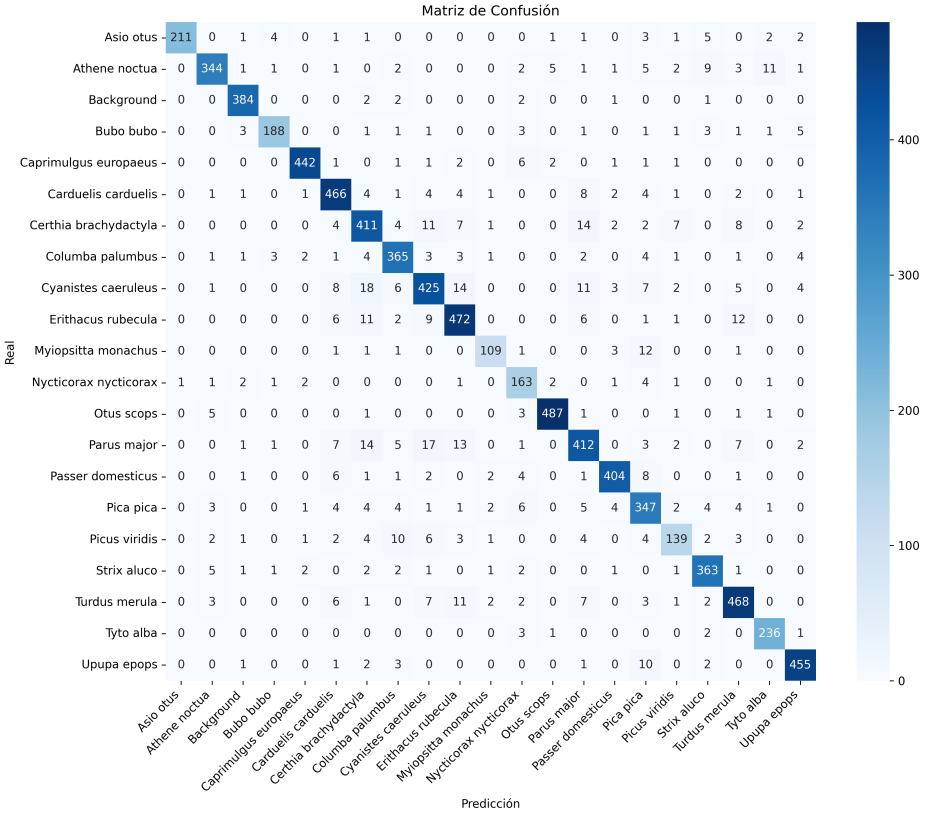

Figura 5: Mejor matriz de confusión, perteneciente al individuo construido con EfficientNetV2B0.

Tabla 2: Resultados detallados por individuo: backbone, capas densas, tasa de dropout, F1-score, parámetros, FLOPs y tiempo por época.

Backbone	**Densas**	**Dropout**	**F1 (%)**	**Params.**	**FLOPs**	**T/época (s)**
	[512, 256]	[0.2, 0.0]	85.05	1.37M	0.12G	48.15
MobileNetV3Small	[512, 512]	[0.0, 0.0]	83.45	1.51M	0.12G	48.28
	[256, 256]	**[0.2, 0.0]**	**84.69**	**1.16M**	**0.12G**	**47.60**
	[256, 256]	[0.0, 0.0]	83.82	3.31M	0.44G	51.07
MobileNetV3Large	[512, 512]	[0.0, 0.0]	84.71	3.76M	0.45G	49.68
	[512]	**[0.2]**	**86.89**	**3.50M**	**0.45G**	**47.71**
	[512, 256]	[0.2, 0.0]	84.80	4.84M	0.80G	51.43
EfficientNetB0	**[512, 512]**	**[0.2, 0.0]**	**85.88**	**4.98M**	**0.80G**	**52.61**
	[256]	[0.4]	81.72	4.38M	0.80G	49.70
	[512, 128]	[0.2, 0.2]	85.20	6.64M	1.46G	51.71
EfficientNetV2B0	[512]	[0.0]	84.49	6.59M	1.46G	49.98
	[512, 64]	**[0.2, 0.2]**	**85.16**	**6.61M**	**1.46G**	**50.51**
	[512, 256]	[0.0, 0.0]	84.16	7.72M	2.43G	53.79
EfficientNetV2B1	**[512]**	**[0.2]**	**83.21**	**7.60M**	**2.43G**	**53.04**
	[512]	[0.3]	82.85	7.60M	2.43G	53.64
	[512, 512]	**[0.4, 0.3]**	**79.88**	**9.76M**	**3.44G**	**56.37**
EfficientNetV2B2	[256, 256]	[0.2, 0.0]	79.94	9.20M	3.44G	55.39
	[512, 256]	[0.0, 0.0]	81.63	9.63M	3.44G	55.42
	[512, 64]	**[0.2, 0.0]**	**81.43**	**11.61M**	**3.77G**	**68.90**
EfficientNetB3	[512]	[0.0]	80.97	11.58M	3.77G	67.96
	[256, 256]	[0.2, 0.0]	81.32	11.25M	3.76G	68.57
	[512]	[0.0]	82.33	13.73M	6.09G	60.11
EfficientNetV2B3	[512, 512]	[0.0, 0.0]	83.54	13.99M	6.09G	60.11
	[512, 128]	**[0.3, 0.0]**	**81.13**	**13.79M**	**6.09G**	**59.68**
	[512, 512]	[0.0, 0.0]	71.34	28.49M	8.78G	50.67
ConvNeXtTiny	**[512, 512]**	**[0.2, 0.0]**	**70.83**	**28.49M**	**8.78G**	**50.48**
	[512]	[0.0]	69.02	28.23M	8.78G	51.00

Tabla 3: Resultados y métricas de entrenamiento para cada individuo.

Backbone	Densas	Dropout	LR	WD	T total (min)	Épocas	T/época (s)	F1 Train	F1 Val
MobileNetV3Small	**[256,256]**	**[0.2, 0.0]**	1e-3	1e-4	51.30	37	83.19	97.90	90.16
MobileNetV3Large	[512]	[0.2]	1e-3	1e-4	43.34	31	83.89	99.68	90.89
EfficientNetB0	[512, 512]	[0.2, 0.0]	3e-4	1e-5	57.30	40	85.95	96.39	91.32
EfficientNetV2B0	**[512, 64]**	**[0.2, 0.2]**	3e-4	1e-5	62.63	44	85.40	93.54	91.73
EfficientNetV2B1	[512]	[0.2]	2e-4	1e-5	81.89	54	90.99	90.78	89.98
EfficientNetV2B2	[512, 512]	[0.4, 0.3]	1e-4	1e-5	82.34	52	95.01	89.47	87.61
EfficientNetB3	[512, 64]	[0.2, 0.0]	1e-4	1e-5	105.78	53	119.75	91.74	90.45
EfficientNetV2B3	**[512, 128]**	**[0.3, 0.0]**	1e-4	1e-5	73.72	43	102.87	90.82	90.24
ConvNeXtTiny	[512, 512]	[0.2, 0.0]	1e-4	1e-2	73.05	51	85.94	95.94	87.31

Tabla 4: Métricas de los individuos finales y resultados sobre la evaluación en el conjunto de prueba. Estructura, parámetros, FLOPs, *F1 score*, precisión, sensibilidad, exactitud balanceada, *top-3 score* y clase peor representada.

Backbone	Densas	Dropout	Params.	FLOPs	F1 macro	Prec. macro	Recall macro	Bal. Acc.	Top-3	Recall mín	Clase mín
MobileNetV3Small	**[256,256]**	**[0.2, 0.0]**	1.16M	0.12G	89.87	90.08	89.72	89.72	97.02	81.32	Picus Viridis
MobileNetV3Large	[512]	[0.2]	3.50M	0.45G	90.83	91.24	90.57	90.57	97.46	74.18	Picus viridis
EfficientNetB0	[512, 512]	[0.2, 0.0]	4.98M	0.80G	84.27	86.21	83.94	84.42	93.88	63.21	Myiopsitta monachus
EfficientNetV2B0	**[512, 64]**	**[0.2, 0.2]**	6.61M	1.46G	90.90	91.21	90.71	90.71	97.36	76.37	Picus viridis
EfficientNetV2B1	[512]	[0.2]	7.60M	2.43G	90.03	90.37	89.80	89.80	97.23	74.73	Picus viridis
EfficientNetV2B2	[512, 512]	[0.4, 0.3]	9.76M	3.44G	88.99	89.30	88.80	88.80	96.81	74.73	Picus viridis
EfficientNetB3	[512, 64]	[0.2, 0.0]	11.61M	3.77G	86.20	87.33	85.82	85.82	94.90	68.57	Myiopsitta monachus
EfficientNetV2B3	**[512, 128]**	**[0.3, 0.0]**	13.79M	6.09G	89.64	89.49	89.97	89.97	96.87	78.77	Cyanistes caeruleus
ConvNeXtTiny	[512, 512]	[0.2, 0.0]	28.49M	8.78G	85.69	85.92	85.61	85.61	95.88	67.03	Picus viridis

4.2. Despliegue en el sistema embebido: NVIDIA Jetson Nano (2 GB)

En esta iteración del proyecto, con solo 20 clases, todas las arquitecturas presentan un muy buen desempeño; sin embargo, se espera que según aumente la complejidad del conjunto de datos, los individuos más ligeros ofrezcan peores resultados. Es por esto por lo que se han seleccionado 3 individuos, el más ligero, el construido sobre MobileNetV3Small; uno intermedio, el construido sobre EfficientNetV2B0, y uno grande, el construido sobre EfficientNetV2B3 [10] Las arquitecturas seleccionadas para el despliegue se encuentran resaltadas en las Tablas 3 y 4. Así partiremos de unas estimaciones del rendimiento en nuestro sistema empotrado para diferente capacidad computacional, lo que puede ser de utilidad en un futuro.

Una vez seleccionadas y desplegadas las arquitecturas, se evaluó el impacto de reducir la precisión mediante cast o cuantización. Se observó que la cuantización a `int8` ofrecía peores resultados que la conversión a `fp16`. Después, se compararon los rendimientos de los modelos bajo los dos diferentes regímenes de potencia que ofrece la placa. Por último, se sometió a la placa a un entorno similar al de despliegue para observar si se comprometían los resultados debido a las altas temperaturas.

Cabe mencionar que no se considera explícitamente el uso de CPU y GPU en inferencia porque, con independencia del individuo y régimen de trabajo, y bajo máxima carga computacional, el uso de CPU es marginal mientras que el de GPU es máximo. Además, un denominador común en todas las mediciones es el uso de RAM del SO, unos 410 MB en promedio y la RAM que reserva TensorRT para compilar cada modelo, en torno a 1,0 GB, aunque varía levemente con cada modelo compilado. El máximo de memoria RAM accesible por el usuario es 1980 MB.

4.2.1. Decremento de rendimiento computacional en modelos cuantizados

Una de las características más importantes de este sistema es la aceleración por hardware. De aquí surge uno de los resultados más contraintuitivos, a priori, de los que hemos encontrado a lo largo de la memoria. Hemos observado que los modelos cuantizados a `int8` no ofrecen mejoras de velocidad respecto a los modelos con precisión mixta. Esto ha motivado una investigación sobre las razones que hay detrás de este resultado.

La Jetson Nano de 2 GB utiliza una GPU Tegra X1 basada en la arquitectura Maxwell, la cual carece de unidades de hardware optimizadas para aritmética de enteros de baja precisión como es `int8`, a diferencia de placas más actuales. Esto conlleva a que estas operaciones estén emuladas. Esta falta de soporte nativo hace que el coste computacional de la emulación compense con creces los beneficios de la cuantización.

No obstante, el cálculo en `fp16` tiene soporte nativo en hardware debido a que Maxwell permite el almacenamiento de datos con esta precisión, lo que permite que los kernels de `fp16` como los proporcionados por las librerías cuBLAS y cuDNN aprovechen el menor ancho de banda de esta precisión para agilizar los cálculos.

En resumen, respaldándonos en los foros de NVIDIA, encontramos que, bajo la ausencia de soporte nativo para `int8`, nuestra mejor opción para agilizar los cálculos es `fp16`.

[10] Se ha preferido esta estructura frente a su versión inicial debido a que presentan un menor tiempo de entrenamiento que sus predecesores y son la opción a elegir pensando en el futuro. Esta decisión viene respaldada por los creadores de estas arquitecturas, Tan y V. Le (2021) [14].

4.2.2. Estudio de la optimización de TensorRT habilitando `fp16` (5W)

El encargado de realizar las optimizaciones para la aceleración de la red neuronal es TensorRT. Esto lo consigue mediante varias técnicas: fusión de capas contiguas, conversión de precisión, auto-ajuste de kernels y simplificación del grafo. El objetivo es la minimización de la latencia manteniendo la exactitud.

Individuo	**Prec.**	**F1 (%)**	**RAM (MB)**	**FPS (1/s)**	**Lat. (ms)**	**Tamaño (MB)**
MobileNetV3Small	`fp32`	88.75	1542	108.36	9.22	5.12
[256, 256], [0,2, 0,0]	`fp16`	88.75	1547	129.02	7.34	2.96
EfficientNetV2B0	`fp32`	90.94	1596	27.52	36.33	27.44
[512, 64], [0,2, 0,2]	`fp16`	90.76	1578	33.13	30.17	14.43
EfficientNetV2B3	`fp32`	88.80	1655	9.46	105.72	56.96
[512, 128], [0,3, 0,0]	`fp16`	88.79	1617	12.14	82.33	30.14

Tabla 5: Métricas en régimen de ahorro de energía (5 W) tomados a una temperatura de 28° para los diferentes modelos.

Se observa que en todos los casos *F1* se mantiene, prácticamente, constante, pero las inferencias por segundo y la latencia bajan considerablemente.

4.2.3. Comparación de métricas para los dos perfiles de rendimiento de la placa

Esta placa presenta dos perfiles de rendimiento: bajo consumo, con un límite de consumo de 5 W, y máximo rendimiento, con un consumo límite de 10 W. En este apartado se han estudiado los desempeños de las versiones de los individuos con precisión mixta.

Individuo	**Potencia (W)**	**RAM (MB)**	**FPS (1/s)**	**Lat. (ms)**	**ΔT** (ºC)
MobileNetV3Small	10	1549	173.17	5.76	1.75
[256, 256], [0,2, 0,0]	5	1547	129.02	7.74	0.50
EfficientNetV2B0	10	1578	47.49	21.05	3.25
[512, 64], [0,2, 0,2]	5	1578	33.13	30.17	1.25
EfficientNetV2B3	10	1620	17.50	57.14	5.00
[512, 128], [0,3, 0,0]	5	1617	12.14	82.32	1.00

Tabla 6: Métricas en los regímenes de potencia: ahorro de energía (5 W) y máximo rendimiento (10 W), tomados a una temperatura de 28° para los diferentes modelos en `fp16`.

Observamos un aumento muy notorio del rendimiento a cambio de un mayor consumo de la placa y un mayor calentamiento de la misma. Se puede apreciar, en el caso de 10 W, que el incremento de temperatura aumenta claramente con el incremento de complejidad computacional; sin embargo, en el caso de 5 W, la diferencia en los incrementos de temperatura es menor y pueden encontrarse fluctuaciones respecto a la tendencia observada en el caso anterior.

4.2.4. Robustez frente a temperatura de las métricas

Para este estudio se evaluaron modelos en dos escenarios, en el primero de ellos, un ventilador enfría el disipador de la placa manteniéndolo, prácticamente, a temperatura ambiente; en el otro caso, se coloca la placa dentro de un recipiente sin refrigeración y se aumenta la temperatura

del mismo considerablemente, sometiéndolo a cargas altas de trabajo hasta que se estabiliza la temperatura. Ambos escenarios fueron estudiados en el modo de bajo consumo.

Por otro lado, según la documentación interna presente en la placa [11], los protocolos frente a la temperatura en la GPU se activan a los 15°C mediante refrigeración activa y crecen en agresividad con la temperatura; en especial, a partir de los 70° C comienzan a apreciarse pérdidas importantes de rendimiento por *throttle*. A partir de este punto, el sistema comienza a estar muy restringido llegando a apagarse de ser necesario al superar los 100° C.

En resumen, no se deberían apreciar pérdidas importantes de rendimiento por debajo de los 65° C.

Individuo	$\mathbf{T}_{\max}$ (°C)	**RAM (MB)**	**FPS (1/s)**	**Lat. (ms)**
MobileNetV3Small	28.50	1547	129.02	7.74
[256, 256], [0,2, 0,0]	50.75	1549	126.65	7.88
EfficientNetV2B0	29.25	1578	33.13	30.17
[512, 64], [0,2, 0,2]	51.25	1577	33.08	30.07
EfficientNetV2B3	29.00	1617	12.14	82.32
[512, 128], [0,3, 0,0]	53.00	1618	12.14	82.38

Tabla 7: Métricas en régimen de ahorro de energía (5 W) partiendo de 28 °C y de 50 °C, para `fp16`.

Se observa que, en ausencia de refrigeración, las métricas son prácticamente análogas al caso con refrigeración. Si hay políticas de *thermal throttle* pasivas, son muy ligeras.

5. Trabajo futuro

5.1. Base de datos

Como ya se ha mencionado en la sección anterior, un filtrado más fino, que tuviese en cuenta la posibilidad de etiquetado erróneo entre especies, podría mejorar considerablemente los resultados. En este trabajo ya se intentó la implementación del modelo Google Voice Classifier, pero era lento, demasiado discriminatorio con las muestras que teníamos y no filtraba del todo bien el ruido, por lo que se prefirió hacer un clasificador propio que no contemplase este aspecto. Por ejemplo, podría reutilizarse el individuo más ligero para este fin. Además, en esta línea de pensamiento, podría resultar interesante tener N modelos pequeños y robustos especializados en subconjuntos diferentes de clases para el filtrado de un conjunto de datos de, por ejemplo, $N \cdot 20$ clases.

Así, con un mejor filtrado y con una mayor complejidad, estaríamos cerca de tener una alternativa seria frente al fototrampeo.

5.2. Algoritmo genético

El enfoque inicial fue tener una guía del desempeño de diferentes modelos según su coste computacional de cara a un estudio en múltiples placas. Pero, como se ha visto, un estudio tan grande de arquitecturas preentrenadas no ha resultado beneficioso; ya que, para cubrir esta exigencia, se

[11] ls /sys/class/thermal/ y ls /boot/dtb/ nos dan los directorios que nos interesan. En el primer caso, encontramos diferentes subdirectorios relevantes que atañen a cada objeto con una estrategia sobre la subida de temperaturas: /thermal_zoneX. Dentro de ella estan los diferentes límites de temperatura para los que se adopta una estrategia, trip_point_X_temp. El tipo de estos puntos (trip_point_X_type) nos marca la estrategia que se sigue: refrigeración activa mediante pwm, refrigeración pasiva mediante *throttle* o apagado por prevención. En el segundo de los casos, estamos interesados en el binario de bpmp, donde se detallan las estrategias.

han introducido sesgos importantes a favor de las arquitecturas más rápidas, que no han podido ser mitigados. Entonces, una nueva aproximación debería considerar *backbones* de manera individual, o grupos menos diversos, donde sea coherente la implementación de una función de coste multiobjetivo y puedan encontrarse mejores configuraciones.

Además, como se mencionó, *Early Stopping* en lugar de entrenamientos a 5 épocas debería ser mucho mejor enfoque.

Por último, quizás pueda ser interesante plantear este problema de optimización de la mejor arquitectura con *transfer learning* respecto a la complejidad de la base de datos. Creando bases de datos más largas, incluyendo subespecies, etc.

5.3. Despliegue en sistemas embebidos

El paso natural, como ya se comentó en la introducción, es desarrollar el resto del sistema de detección: configurar un micrófono para que mande periódicamente grabaciones a la Jetson Nano, procesar estas grabaciones, analizarlas y mapear los resultados, como se hace en [5]. Por otro lado, en el peor de los casos, donde tuviésemos que utilizar el individuo más pesado, podríamos pensar en liberar RAM mediante una mejor gestión de la partición *swap* y así tener un mayor margen disponible.

Referencias

[1] Frederik P. Brinck and contributors. Audiomentations. https://github.com/iver56/audiomentations (accedido en marzo de 2025).

[2] François Chollet. Building powerful image classification models using very little data, 2016. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html (accedido en marzo de 2025).

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[4] Freesound. https://freesound.org (accedido en febrero de 2025).

[5] Jonas Höchst, Hicham Bellafkir, Patrick Lampe, Markus Vogelbacher, Markus Mühling, Daniel Schneider, Kim Lindner, Sascha Rösner, Dana G. Schabo, Nina Farwig, and Bernd Freisleben. Bird@Edge: Bird Species Recognition at the Edge. In *International Conference on Networked Systems (NETYS)*. Springer, May 2022.

[6] Lihang Hong. Domain adaptation for birdcall recognition: Progressive knowledge distillation with semi-supervised and self-supervised soundscape labeling. In *Notebook for the BirdCLEF Lab at CLEF 2024*, volume 3740 of *CEUR Workshop Proceedings*, page 198, 2024.

[7] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[8] Arunodhayan Sampath Kumar, Tobias Schlosser, Stefan Kahl, and Danny Kowerko. Improving learning-based birdsong classification by utilizing combined audio augmentation strategies. *Ecological Informatics*, 82:102699, 2024.

[9] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.

[10] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018.

[11] Loris Nanni, Gianluca Maguolo, and Michelangelo Paci. Data augmentation approaches for improving animal audio classification. *Ecological Informatics*, 57:101084, 2020.

[12] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search, 2019.

[13] Justin Salamon and Juan Pablo Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal Processing Letters*, 24(3):279–283, March 2017.

[14] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training, 2021.

[15] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers, 2023.

[16] xeno-canto: bird sounds from around the world. https://www.xeno-canto.org (accedido en febrero de 2025).

[17] Zhao Zhao, Sai hua Zhang, Zhi yong Xu, Kristen Bellisario, Nian hua Dai, Hichem Omrani, and Bryan C. Pijanowski. Automated bird acoustic event detection and robust species classification. *Ecological Informatics*, 39:99–108, 2017.