SISTEMAS INTELIGENTES PARA LA GESTIÓN DE LA EMPRESA

MÁSTER EN INGENIERÍA INFORMÁTICA

Deep Learning para clasificación

Práctica 2 - Clasificación de especies de pájaros

Autores
Adriano García-Giralda Milena
Sergio Hervás Cobo

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Mayo de 2024

Índice

1. Introducción

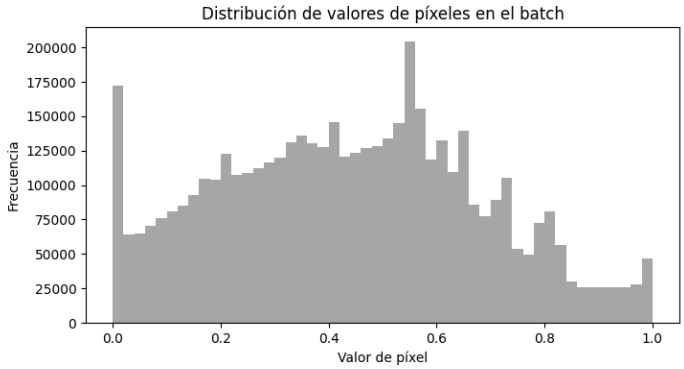
El reconocimiento y clasificación de imágenes es una de las tareas más importantes en el aprendizaje automático y la visión por computadora. Dentro de este campo, la clasificación fina plantea un desafío mayor, ya que las categorías presentan diferencias muy sutiles que requieren un análisis más detallado.

El conjunto de datos CUB-200-2011 es un estándar para este tipo de problemas: contiene 11 788 imágenes de 200 especies de aves, con variaciones en iluminación, fondos y posturas. Además de las imágenes, incluye atributos sobre las partes anatómicas, lo que permite enfoques más precisos y explicables.

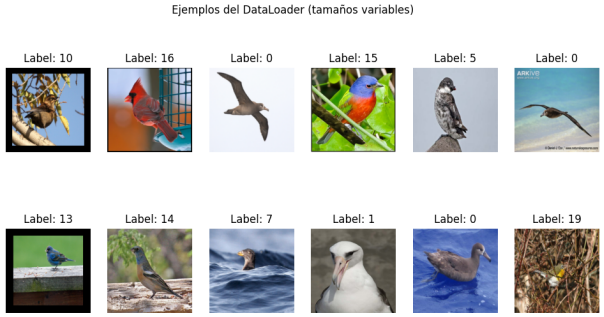
El objetivo de esta práctica es entrenar un modelo de aprendizaje profundo capaz de identificar correctamente la especie de un ave a partir de su fotografía. Para ello exploraremos distintas estrategias de entrenamiento para las redes neuronales. Más allá de la precisión numérica (precisión global, matriz de confusión), evaluaremos la interpretabilidad del modelo mediante mapas de activación y análisis de atención.

1.1. Visualización

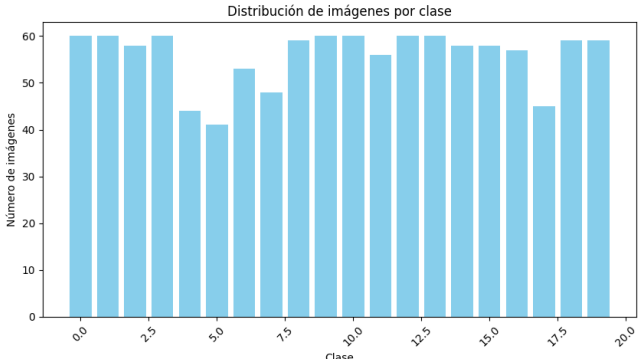
En primer lugar exploramos de manera visual el conjunto de datos para identificar los tipos de imágenes y atributos con los que trabajaremos. Generamos un histograma de distribución de intensidades de píxel y examinamos posibles indicios de ruido. Además, representamos la frecuencia de aparición de cada clase para evaluar su equilibrio.



(a) Histograma de intensidades de píxel



(b) Ejemplos representativos



(c) Número de imágenes por clase

De este análisis extraemos las siguientes conclusiones:

- El conjunto de datos no presenta ruido apreciable a simple vista.
- La distribución de imágenes por categoría es casi homogénea, aunque no perfectamente equilibrada.
- Cada clase dispone de un número relativamente bajo de muestras (aproximadamente 45–60 imágenes por categoría, según los subconjuntos $\times 20$ y $\times 200$).
- El histograma de intensidades indica una tendencia hacia tonalidades más oscuras en la mayoría de las imágenes.

En cuanto a los atributos, cada uno se codifica como `propiedad_valor` (por ejemplo, `has_bill_shape::dagger`). Estos metadatos resultan fundamentales para diseñar nuestro modelo multimodal, pues permiten incorporar información semántica que complementa la señal visual.

2. Preparación del conjunto de datos

En esta sección describimos de manera concisa el flujo completo de preparación de datos, sin incluir fragmentos de código, para centrar la atención en la lógica y los objetivos de cada etapa:

1. **Definición de transformaciones**
   - Transformación "estándar" que redimensiona cada imagen a 224×224 píxeles y la convierte en tensor.
   - Tres operaciones de aumento de datos independientes:
     - Espejado horizontal (`flip`).
     - Rotación aleatoria limitada a $\pm15°$.
     - Zoom aleatorio en un rango $[0.8, 1.2]$ mediante transformación afín.
2. **Estrategias de data augmentation**
   - *Aumento sencillo (duplicado)*: por cada imagen original se genera una única variante aleatoria aplicando solo una de las tres operaciones anteriores, duplicando así el tamaño del conjunto de entrenamiento sin combinar transformaciones.
   - *Aumento multimodal (cuadriplicado)*: por cada imagen original se crean tres variantes, una para cada operación, obteniendo cuatro ejemplares por muestra. Esta estrategia mejora la variabilidad pero resultó demasiado costosa en tiempo y memoria, por lo que no se empleó en el estudio del fine tuning y solo se planteó para el modelo multimodal final.
3. **Carga y partición del dataset**
   - Se carga el directorio principal de imágenes con un loader genérico (p. ej., `ImageFolder`), aplicando el transform estándar.
   - Se divide de forma reproducible en entrenamiento y validación (80 % / 20 %) utilizando una semilla fija para garantizar comparaciones consistentes.
4. **Construcción del conjunto de entrenamiento**
   - Se instancia un segundo loader sobre la misma carpeta, usando la transformación de aumento sencillo.
   - A partir de la partición de entrenamiento, se extrae un subconjunto de índices en el loader aumentado.
   - Se concatenan ambos subconjuntos (original y aumentado) para formar el conjunto definitivo de entrenamiento, que se recorre con barajado aleatorio en minibatches.

5. **Pipeline multimodal**

   - Para incorporar atributos de aves (forma del pico, presencia de manchas, etc.), se define un dataset que, además de devolver la imagen, construye un vector binario de atributos a partir de tres ficheros de metadatos:
     a) `attributes.txt`: lista de nombres de atributos (por ejemplo `has_bill_shape::dagger`), a partir de la cual se crea un índice para cada propiedad.
     b) `images.txt`: mapea cada nombre de archivo de imagen a un identificador numérico único (`img_id`).
     c) `image_attribute_labels.txt`: contiene líneas con formato `<img_id><attr_id><is_localized><is_present><confidence>`, donde `is_present` indica si ese atributo está presente en la imagen.

     A partir de estos datos, para cada muestra se genera un vector de dimensión igual al número total de atributos, inicializado a cero y al que se ponen a 1 las posiciones correspondientes a los atributos presentes según `image_attribute_labels.txt`.
   - Durante el entrenamiento multimodal, se aplica la misma lógica de data augmentation a las imágenes, mientras que estos vectores de atributos binarios permanecen inalterados para conservar la información semántica.

6. **Preparación de los DataLoaders**

   - Un `DataLoader` de entrenamiento recorre el conjunto combinado (imagen + aumentos) con tamaño de lote configurable y barajado activo.
   - Otro `DataLoader` de validación evalúa únicamente sobre las muestras limpias de validación, sin barajado.

7. **Collate personalizado**

   - En cada minibatch se agrupan y apilan las imágenes, los vectores de atributos (en el caso multimodal) y las etiquetas de clase en tensores listos para alimentar el modelo.

3. Entrenamiento y Modelos

Antes de describir los detalles de los bucles de entrenamiento y las arquitecturas evaluadas, presentamos brevemente los pilares metodológicos que sustentan nuestro enfoque y las arquitecturas propuestas:

3.1. Arquitecturas no multimodales

CustomCNN Definida en `customcnn.py`, consta de:

- Tres bloques convolutivos: `Conv2d(3,32,3,padding=1)` → `BatchNorm2d(32)` → `ReLU` → `MaxPool2d(2)` sucesivamente con canales (32→64, 64→128).
- `AdaptiveAvgPool2d((1,1))` para obtener un vector de 128 características.
- Clasificador: `Flatten` → `Linear(128,256)` → `ReLU` → `Dropout(0.5)` → `Linear(256,`*`num_classes`*`)`.

ResNet-50 Utilizada como backbone preentrenado en ImageNet, con pesos inicialmente congelados.

- Se reemplaza la cabeza original (`fc`) por: `Linear(in_features,512)` → `ReLU` → `Dropout(0.5)` → `Linear(512,`*`num_classes`*`)`.
- Se descongelan todas las capas de `layer4` y la nueva cabeza para permitir fine-tuning parcial.

EfficientNet-B4 Backbone preentrenado con estrategia similar:

- Pesos iniciales congelados; se descongelan los últimos tres bloques MB-Conv y toda la cabeza original.
- Se sustituye la cabeza por: `Dropout(p=0.5)` → `Linear(in_features,`*`num_classes`*`)`.

3.2. Arquitectura multimodal (MultiModalResNet)

Basada en ResNet-50 preentrenado, fusiona imágenes con atributos semánticos:

1. **Imagen:** extracción de características con ResNet-50 hasta `layer4`, pesos congelados salvo en `layer4`. La capa `fc` se reemplaza por `Identity` para obtener un vector de N dimensiones.
2. **Atributos:** proyección de un vector binario de D atributos mediante `Linear(D,256)` → `ReLU` → `Dropout(0.5)`.
3. **Fusión:** concatenación de ambos vectores ($N+256$) y clasificación con `Linear(N+256,512)` → `ReLU` → `Dropout(0.5)` → `Linear(512,`*`num_classes`*`)`.

3.3. Transferencia de aprendizaje

Debido a las limitaciones de nuestros modelos personalizados, que carecían de riqueza a la hora de poder extraer características con datos tan limitados, recurrimos a redes preentrenadas (ResNet-50 y EfficientNet-B4) como extractores de características. Durante la fase inicial, sus pesos permanecieron congelados para preservar los patrones genéricos ya aprendidos; únicamente se reentrenó la capa final, sustituida por un clasificador específico compuesto por una capa lineal, activación ReLU y, en el caso de ResNet, un `Dropout` para mitigar el sobreajuste. Esta estrategia acelera el entrenamiento y mejora el rendimiento al aprovechar el conocimiento previo de los modelos.

3.4. Logging con Weights & Biases

Para monitorizar y comparar experimentos de forma sistemática, integramos Weights & Biases (W&B). Con W&B registramos automáticamente hiperparámetros (tasa de aprendizaje, tamaño de lote, optimizador, ...) y métricas clave (pérdida, precisión en entrenamiento y validación). Su panel interactivo facilita la detección de tendencias, problemas de sobreajuste y el efecto de ajustes de hiperparámetros.

3.5. Ajuste automático de hiperparámetros

Utilizamos Optuna para explorar de forma eficiente el espacio de:

- *Learning rate*: 10^{-5}–10^{-3} (escala logarítmica).
- *Batch size*: $\{16, 32, 64\}$.
- *Optimizers*: Adam o SGD (momentum 0.8–0.99).

Optuna emplea un muestreador TPE y pruning temprano (MedianPruner) para descartar rápidamente configuraciones poco prometedoras. La mejor combinación se selecciona automáticamente y se usa para entrenar el modelo final.

3.6. Enfoque multimodal

Nuestro modelo multimodal fusiona dos fuentes de información:

- *Visual*: extracción de características jerárquicas mediante un backbone preentrenado (ResNet-50).
- *Semántica*: vector binario de atributos biológicos (forma de pico, patrones de plumaje, ...) obtenido de los metadatos de CUB-200-2011.

Estas representaciones se concatenan en una cabeza fully-connected, permitiendo al modelo aprender correlaciones entre rasgos visuales y descriptores semánticos, mejorando tanto la precisión como la interpretabilidad.

3.7. Explicabilidad

Para entender las decisiones del modelo, aplicamos dos técnicas:

- *Grad-CAM*: genera mapas de activación en la última capa convolucional (layer4 de ResNet), resaltando regiones de la imagen que influyen en la predicción.
- *LIME*: identifica superpíxeles determinantes mediante aproximaciones locales interpretables, mostrando cómo detalles globales y locales contribuyen a la clasificación.

3.8. Entrenamiento Convencional

Para el entrenamiento de los modelos utilizamos la función `train_model()`, que implementa el bucle estándar de *train/validation* junto con varias mejoras:

- Inicialización opcional de un `wandb` run para monitorizar métricas, hiperparámetros y pesos durante el entrenamiento.
- Uso de `CrossEntropyLoss` como función de coste.
- Selección de optimizador configurable entre ADAM y SGD (con momentum).
- Scheduler `ReduceLROnPlateau` que reduce la tasa de aprendizaje si la precisión de validación no mejora tras un número de épocas ('patience') determinado.
- Detección de la mejor precisión de validación y guardado del estado del modelo cuando ésta mejora.
- Early stopping tras un número prefijado de reducciones de learning rate sin avance en la métrica de validación.
- Registro en consola y opcionalmente en `wandb` de la pérdida de entrenamiento, precisión de validación y tasa de aprendizaje en cada época.

El flujo es:

1. **Entrenamiento (`model.train()`)**: para cada batch se calcula la salida, la pérdida, se retropropaga y actualizan parámetros.

2. **Validación (`model.eval()`)**: se evalúa precisión sin gradientes para medir desempeño real.
3. **Scheduler.step**: ajusta el learning rate según la precisión de validación.
4. **Checkpoints y early stopping**: se guarda el mejor modelo y se detiene el bucle si no hay mejoras tras varias reducciones de LR.

3.9. Entrenamiento Multimodal

Para el modelo `MultiModalResNet` (que fusiona características visuales de un ResNet-50 preentrenado con un embedding de atributos semánticos), la rutina de entrenamiento es idéntica, salvo que:

- En el `forward` se recibe un par (imagen, atributos), se extraen features de imagen (solo `layer4` de ResNet permanece entrenable) y se proyectan los atributos mediante una capa lineal con `Dropout`.
- Ambos vectores se concatenan y se envían a una cabeza fully-connected para obtener la predicción final.
- En los `DataLoader` multimodales, cada batch contiene tripletes (imgs, attrs, labels) y se utiliza un `collate_fn` personalizado que apila tanto imágenes como vectores de atributos en tensores adecuados.

3.10. Búsqueda de Hiperparámetros

Para optimizar {learning_rate, batch_size, optimizer} empleamos dos estrategias:

Grid Search clásico: iteramos combinaciones discretas de tasas de aprendizaje, tamaños de lote y optimizadores, cada experimento registrado en `wandb`, y guardamos la mejor configuración según la precisión de validación.

Optuna: definimos un *study* con búsqueda bayesiana adaptativa y pruning de trials lentos, permitiendo explorar rangos continuos (por ejemplo, $\text{lr} \in [10^{-5}, 10^{-3}]$) y detener automáticamente aquellos ensayos que no prometen mejoras.

Una vez encontrada la mejor configuración, volvemos a entrenar el modelo completo con esos hiperparámetros y guardamos el peso final.

Con este esquema cubrimos tanto el entrenamiento tradicional de modelos puramente convolutivos como el de arquitecturas multimodales y la optimización sistemática de sus hiperparámetros.

4. Discusión de los resultados

Para ajustar los hiperparámetros realizamos un total de 84 experimentos (7 de ellos cancelados), centrados en fine-tuning de ResNet-50 y EfficientNet-B4, así como en el modelo multimodal basado en ResNet-50. La Figura 2 muestra el número de ejecuciones realizadas.

clasification	8 minutes ago	Team	24
optuna-tuning-clasificacion	1 hour ago	Team	67

Figura 2: Número de runs ejecutados y cancelados durante la búsqueda de hiperparámetros.

Los mejores valores de learning rate hallados fueron:

$$\mathrm{lr}_{\mathrm{ResNet50}} = 6.47 \times 10^{-5}, \qquad \mathrm{lr}_{\mathrm{EfficientNet\text{-}B4}} = 1 \times 10^{-3}.$$

Cuadro 1: Comparativa de mejores resultados precisión (%) en los subconjuntos $\times 20$ y $\times 200$ clases

Modelo	**x20**	**x200**
ResNet-50	**89.23**	**71.30**
Multimodal (ResNet + atributos)	87.44	66.82
EfficientNet-B4	85.20	68.45
CustomCNN	30.04	–

Precisión de validación vs. época

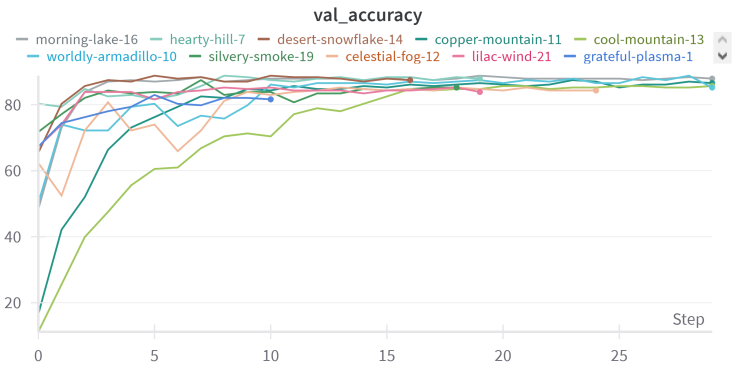


Figura 3: Evolución de la precisión de validación a lo largo de las épocas para los distintos modelos y optimizadores. Captura de WANDB

Tasa de aprendizaje vs. época

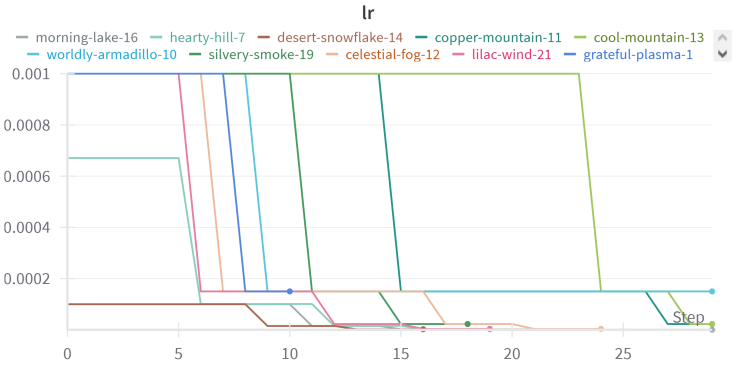
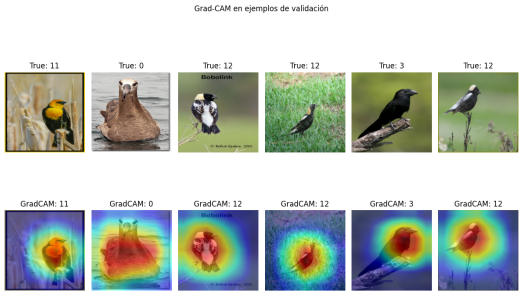
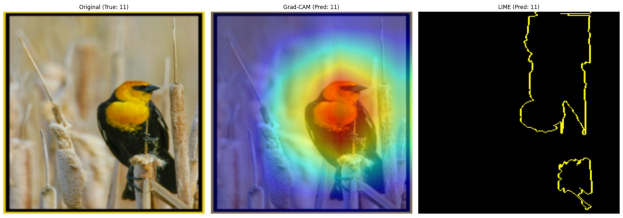


Figura 4: Variación de la tasa de aprendizaje según el scheduler `ReduceLROnPlateau` durante el entrenamiento.

Visualizaciones de explicabilidad



(a) Grad-CAM



(b) LIME

Figura 5: Comparativa de Grad-CAM y LIME en una misma imagen: (a) Grad-CAM destaca con un mapa de calor las zonas críticas. (b) LIME resalta superpíxeles influyentes,

En definitiva, nuestro estudio demuestra que las arquitecturas basadas en transferencia de aprendizaje ofrecen un rendimiento muy superior al entrenamiento desde cero cuando los datos son limitados. De forma concreta:

- **Mejor rendimiento**: ResNet-50 preentrenado, ajustado con fine-tuning y optimizador Adam, alcanzó la mayor precisión en los subconjuntos de 20 y 200 clases.
- **Backbones ligeros**: EfficientNet-B4 presentó un compromiso excelente entre velocidad y exactitud, siendo la segunda mejor opción en ambas particiones.
- **Modelos multimodales**: La fusión de características visuales con atributos semánticos no mejoró la métrica global, sin embargo no se puede destacar que exista otra técnica distinta para dar mayor peso a estas características como el uso de sesgos los cuales nosotros no hemos usado.
- **Limitaciones de modelos desde cero**: CustomCNN, sin preentrenamiento, quedó muy por detrás debido a la escasez de ejemplos por clase.
- **Técnicas de explicabilidad**: Grad-CAM y LIME confirmaron que los modelos aprenden realmente a centrarse en el ave, siendo Grad-CAM especialmente claro y accesible para validar visualmente el proceso de decisión.
- **Importancia del pipeline de datos**: El uso de data augmentation sencillo (duplicado) resultó suficiente para mejorar la generalización sin incurrir en costes excesivos de computación, mientras que el enfoque multimodal mostró cómo integrar metadatos sin complicar el flujo de datos.
- **Automatización y reproducibilidad**: La combinación de W&B para logging y Optuna para búsqueda de hiperparámetros permitió optimizar sistemáticamente tasas de aprendizaje, tamaños de lote y optimizadores, garantizando resultados más consistentes y reproducibles.

En conjunto, estos hallazgos subrayan la efectividad de aprovechar modelos preentrenados y metodologías automatizadas de ajuste en problemas de clasificación complejos, al tiempo que se resalta el valor de la interpretabilidad para asegurar la confianza en entornos de aplicación real.

Referencias

[Akiba et al., 2023] Akiba, T. et al. (2023). Optuna: A hyperparameter optimization framework. `https://optuna.readthedocs.io/en/stable/`. Accedido en mayo de 2025.

[Biases, 2023] Biases, W. . (2023). Weights & biases documentation. `https://docs.wandb.ai/`. Accedido en mayo de 2025.

[Gildenblat et al., 2021] Gildenblat, J. et al. (2021). Pytorch grad-cam: Class activation maps. `https://github.com/jacobgil/pytorch-grad-cam`. Accedido en mayo de 2025.

[Panati et al., 2022] Panati, C., Wagner, S., and Brüggenwirth, S. (2022). Feature relevance evaluation using grad-cam, lime and shap for deep learning sar data classification. In *2022 23rd International Radar Symposium (IRS)*, pages 457–462.

[Ribeiro et al., 2016] Ribeiro, M. T. et al. (2016). Lime: Explaining the predictions of any machine learning classifier. `https://github.com/marcotcr/lime`. Accedido en mayo de 2025.

[wenewone, nd] wenewone (n.d.). Cub-200-2011. `https://www.kaggle.com/datasets/wenewone/cub2002011`. Accedido en mayo de 2025.