Universidad Francisco de Vitoria

Escuela Politécnica Superior


Deep Learning

# Práctica Grupal 1: Computer Vision

*Profesor:*
D° Moisés Martínez Muñoz

*Alumnos:*
Alfredo Robledano Abasolo
Jorge Barcenilla González
Rubén Sierra Serrano
Pedro García Silgo

Martes 1 de abril de 2025

# Resumen

El presente trabajo consiste en desarrollar modelos de *Deep Learning* para la clasificación de fauna marina a partir de imágenes. Para ello, se utilizarán redes neuronales convolucionales profundas (*Convolutional Neural Networks*, CNNs), empleando un *dataset* de acuarios que contiene fotografías de diversas especies, tales como peces, medusas, pingüinos, tiburones, frailecillos, mantarrayas y estrellas de mar.

El objetivo principal es diseñar un *pipeline* que permita identificar y clasificar estas especies con alta precisión. Para lograrlo, se implementarán técnicas avanzadas como el *transfer learning*, *fine tuning* y *data augmentation*, optimizando el rendimiento del modelo mediante el ajuste de hiperparámetros y el uso de arquitecturas modernas. [1]

Además, se evaluará el desempeño del modelo mediante métricas estándar como la precisión, la sensibilidad y la especificidad.



## Palabras Clave

*Deep Learning*, Redes Neuronales Convolucionales, *Transfer Learning*, clasificación

# Índice

# 1. Desarrollo

## 1.1. Descripción del *dataset*

El *dataset* con el que estamos trabajando posee tres carpetas: una para los datos de *train*, otra para los datos de *test* y, por último, una tercera para los datos de *validation*. En estas carpetas se encuentran las imágenes y un documento en formato `.csv` llamado `annotations`, que contiene la información de las imágenes. La siguiente tabla describe los atributos del documento

| **Atributo** | **Tipo de dato** | **Descripción del atributo** |
|---|---|---|
| filename | string | Nombre de la imagen |
| width | int | Anchura de la imagen |
| height | int | Altura de la imagen |
| class | string | Clase de la imagen |
| xmin | int | Coordenadas de la clase detectada en la imagen |
| ymin | | |
| xmax | | |
| ymax | | |

Figura 1: Atributos del *dataset*

En cuanto a la distribución de las imágenes, tenemos 448 imágenes de *train*, 63 imágenes de *test* y 127 imágenes de *validation*. Por tanto, estamos ante un *dataset* bastante limitado, teniendo en cuenta que estamos trabajando con imágenes y que nuestro objetivo es clasificarlas. Esto abre la puerta al uso de técnicas de *data augmentation*, así como al empleo de arquitecturas de red preentrenadas para aplicar *transfer learning*.

Además, existe un notable desbalanceo en la cantidad de imágenes representativas de cada clase, lo cual refuerza la necesidad de implementar una estrategia adecuada de *data augmentation* para mejorar la diversidad del conjunto de datos y mitigar los efectos negativos del desequilibrio en el entrenamiento del modelo.

## 1.2. Organización del *dataset*

Para facilitar el trabajo con el *dataset*, hemos automatizado su descarga, extracción y organización en carpetas según la clase dominante de cada imagen. Para ello, subimos el archivo `.zip` a Google Drive y desarrollamos un código para descargarlo y descomprimirlo utilizando la librería gdown. Este contiene imágenes de 7 clases diferentes agrupadas en carpetas train, test y valid (validation).

A continuación se procede a etiquetar el dataset, aunque el conjunto de datos sugiere una sobrepoblación de peces en las imágenes, que clase de animal marino

predomina en cada imagen se determinará a partir del área y número de veces que aparece una clase.
Se define el siguiente criterio:

$$\text{Dominant Class} = \arg\max_{c \in \mathcal{C}} (\text{area}_c \cdot \text{count}_c) \tag{1}$$

Para una imagen la clase predominante será aquella cuyo producto área por número de veces que aparece sea mayor.La información que permite este cálculo se encuentra en los archivos csv de anotación disponibles en cada una de las carpetas train, test y valid. Estos archivos contienen las asociaciones entre entidad e imagen, y en particular contienen los bounding boxes de los animales identificados en cada imagen.

Para obtener el conteo de una clase en cada imagen se agrupan las anotaciones por imagen y clase, y se aplica el método .size() que nos devuelve el tamaño de las agrupaciones.

Para obtener el área de una clase primero se calcula para cada entidad y posteriormente se agrupa por imagen y clase para aplicar el método .sum() que nos devuelve la suma de áreas de una clase para cada imagen.

Como consideraciones a futuro se podría tener en cuenta si existe superposición de bounding boxes y en su caso adaptar el cálculo de área, para esta práctica por no especificarse un criterio y ser un enfoque simple no se tienen en cuenta las intersecciones.

Se tiene entonces pares (imagen, clase), para guardar este cálculo en forma de organización del repositorio, se crean en los sets de train, test y valid subcarpetas con nombres las etiquetas y se mueven las imágenes a sus respectivas carpetas. El algoritmo que define esta organización pasa por obtener los nuevos paths utilizando los nombres de imágenes y añadiendo las etiquetas correspondientes. El módulo shutil de Python nos permite mover los archivos y con un bloque try-except se evita que se produzcan errores si alguno de los archivos no se encuentra (esperable si ya se han movido a las subcarpetas).

## 1.3. Preprocesamiento

Una vez el dataset se encuentra organizado en subcarpetas de clase dominante (dentro de cada carpeta *train*, *test* y *validation*), se preprocesan las imágenes definiendo una serie de pasos que permiten obtener entradas adecuadas para nuestros modelos.

Para definir estos pasos a modo de pipeline se utiliza un tipo de dato nativo de tensorflow, el tensorflow.data.Dataset, este tipo de objeto es muy conveniente porque permite definir de manera sencilla y secuencial los pasos para transformar los datos en inputs adecuados para modelos de aprendizaje construidos con

tensorflow. Se trata de un objeto por el que pasan y se pueden hacer operaciones con tensores en un formato eficiente.

Como entradas al pipeline se tienen los paths a las imágenes en los cuales están las etiquetas correspondientes (las imágenes han sido organizadas previamente en sus subcarpetas de clase).

Utilizando el objeto Dataset del módulo data de tensorflow se almacena tensores de paths y son estas cadenas binarias las que se convierten en pares (imagen, etiqueta) tras una serie de pasos:

- Se obtienen las etiquetas en formato one-hot encoded: extrayendo del path el nombre de la etiqueta y mapeandolo a un entero utilizando una tabla hash se obtiene el entero asociado a la clase, que finalmente se convierte a formato one-hot. En este punto el pipeline devuelve pares path a la imagen y etiqueta en formato one-hot.
- Se obtienen las imágenes: utilizando las funciones io de tensorflow.
- Redimensionado de las imágenes: Las imágenes anteriores son de tamaño variable, mientras que los modelos que se van a utilizar esperan datos para entrenar con las mismas dimensiones, primero se añade padding para conseguir que todas las imágenes sean 1024x1024 y después se redimensionan para utilizar un tamaño 64x64 como entrada en el caso Pre-train y 224x224 en el caso Fine-tuning.
- Media 0 y desviación 1: Una vez se tienen las imágenes en un tamaño adecuado se escalan sus características, esto es, los valores de los pixels se escalan para que tengan media 0 y desviación 1. Esto se realiza para conseguir una convergencia más rápida.

## 1.4. *Data Augmentation*

Tras el preprocesamiento de las imágenes, con la finalidad de obtener mayor diversidad en el conjunto de datos se han realizado pequeñas transformaciones en las imágenes, que pueden ayudar a que el modelo sobreentrene menos, mejorando su capacidad de generalización.

Para ello, se han aplicado inversiones horizontales, así como rotaciones, zooms y recortes, todo ello se ha llevado a cabo de manera pseudoaleatoria.

## 1.5. *Pre-Train*

En esta sección se han diseñado 3 modelos que cumplan las especificaciones pedidas:

- Al menos 3 capas convolucionales.
- ReLU como función de activación
- softmax para la salida de la última capa.

- Uso de la optimización RMSprop.
- categorical crossentropy como función de pérdida
- RMSprop para minimizar la función de pérdida

Se pide que utilizar 3 configuraciones de tamaños de kernel, tamaños de pooling, número de capas y tasa de aprendizaje para ejecutar el proceso de entrenamiento.

Se han elegido tres tasas de aprendizaje variable dependiente de un parámetro paciencia que reduce el learning rate si no mejora el entrenamiento, de forma que se refine el mínimo de la pérdida cuando la red no mejora.

Pasos realizados:

- Definir callbacks: se define el EarlyStopping y el ReduceLROnPlateau. Lo que permite detener el entrenamiento cuando la red no mejora sobre datos de validación y por otro lado obtenemos un learning rate variable dependiente del parámetro paciencia.
- Definir el modelo: se definen sus capas a partir de los parámetros elegidos.
- Compilar el modelo: se definen las funciones de perdida y optimizador.

Finalmente con un bucle se obtienen los resultados del entrenamiento y se elige el mejor modelo.

## 1.6. *Fine-Tuning*

Para el caso Fine-Tuning solo se utilizan 3 configuraciones de tasas de learning rate variables y la arquitectura del modelo se basa en un modelo preentrenado, en nuestro caso y tras varias pruebas hemos elegido el modelo MobileNetV3Large.

De nuevo con un bucle se obtienen los resultados del entrenamiento y se elige el mejor modelo.

# 2. Conclusiones

## 2.1. Fase 1: Preparación del entorno

Pese a que en esta sección no se ha realizado ningún modelo, es importante destacar que la preparación del entorno ha sido un proceso fundamental para el desarrollo de la práctica.

También es resulta interesante mencionar el proceso de creación de imágenes sintéticas que al final no se usaron. Con el proceso que realizó, se generó una imagen una imagen sintética para cada imagen que tuviera 2 tipos de animales o más en la misma imagen y que la segunda clase no fuera de fish. Siguiendo esta idea solo se generaron unas 24 imágenes nuevas, y debido a la poca aportación de imágenes de clases poco representadas, se decidió no incluirlas en el *dataset* final.

El proceso se llevó a cabo seleccionando las imágenes candidatas y eliminando los píxeles correspondientes a la primera clase predominante. Esto implicaba reemplazar las áreas ocupadas por los animales de dicha clase con el color del fondo original. A partir de esta modificación, se generó una nueva imagen sintética en la que la clase predominante pasó a ser la segunda más representativa de la imagen original.

Siguiendo esta línea, se produjeron aproximadamente 24 imágenes nuevas; sin embargo, dada la escasa producción de imágenes de clases menos frecuentes, se optó por no incluirlas en el *dataset* final.

Como aprendizaje fuera de estas imágenes sintéticas, se ha comprobado la utilidad de usar pipelines de proceso de imágenes, para automatizar el proceso de preparación de datos para el entrenamiento de modelos.

## 2.2. Fase 2: Implementación en modo Pre-Train

En esta seccion se ha implementado una red neuronal convolucional (CNN) y se han ido generando diferentes modelos a partir de esta red. Se ha ido modificando la arquitectura de la red, Teniendo así los siguientes resultados:

| **Modelo** | **Capas** | **LR Factor** | **LR Patience** | **Val Acc** | **Test Acc** |
|---|---|---|---|---|---|
| 1 | 4 | 0.20 | 5 | 0.7480 | 0.8730 |
| 2 | 3 | 0.10 | 3 | 0.7795 | 0.8254 |
| 3 | 5 | 0.30 | 8 | 0.7717 | 0.7460 |

Figura 2: Comparación de Modelos

**Mejor Modelo: #1**

- **Capas Convolucionales:** 4
- **Factor de Reducción LR:** 0.2
- **Paciencia para Reducción LR:** 5
- **Mejor Época:** 18
- **Precisión de Validación:** 0.7480
- **Precisión de Prueba:** 0.8730

Como se observa existe una diferencia entre la precisión de validación y la de prueba. Hay que tener en cuenta sin embargo que la precisión de test (prueba) es tan alta porque las imagenes del train son muy parecidas a las de test, ya que el dataset está hecho con imágenes un mismo video. Eso puede significar que el modelo no generalize fuera de este video o similares.

Como aprendizaje de esta fase, la limitación de calidad de las imágenes no tiene porqué afectar de forma muy negativa al modelo, y por lo tanto es una prueba que puede ser útil realizar en muchos otros casos debido a su bajo coste computacional.

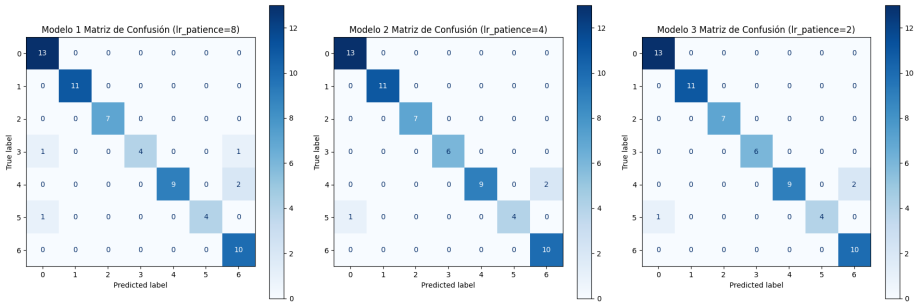


Figura 3: Matrices de confusión.

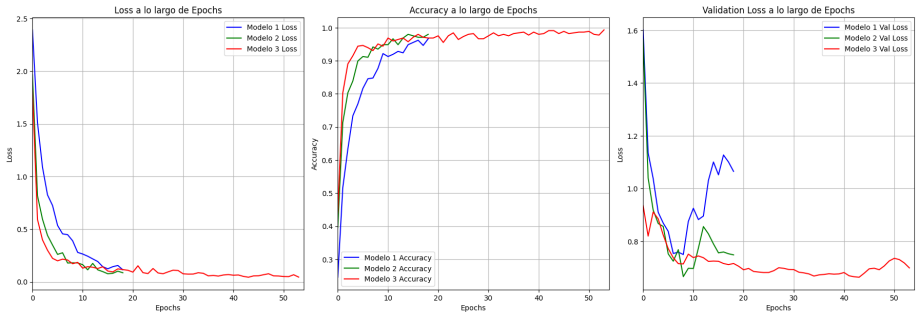


Figura 4: train_loss, accuracy y val_loss

## 2.3. Fase 3: Implementación en modo Fine-Tunning

En esta seccion se ha utilizado un modelo preentrenado como base (MobileNetV3Large) y se han ido generando diferentes modelos modificando la tasa de learning rate variable definido por la paciencia. Para utilizar este modelo se ha eliminado su capa superior y solo se han entrado las 20 últimas capas, para adaptar el aprendizaje a nuestro problema y a la vez aprovechar los pesos preentrenados. Teniendo así los siguientes resultados:

| Modelo | LR Patience | Val Acc | Test Acc |
|---|---|---|---|
| MobileNetV3Large | 8 | 0.8346 | 0.9263 |
| MobileNetV3Large | 4 | 0.8268 | 0.9367 |
| MobileNetV3Large | 2 | 0.8346 | 0.9578 |

Figura 5: Comparación de Modelos

**Mejor Modelo: #3**

- **Paciencia para Reducción LR:** 2
- **Mejor Época:** 36
- **Precisión de Validación:** 0.8346
- **Precisión de Prueba:** 0.9578

Como aprendizaje de esta fase, encontramos la potencia de utilizar un modelo preentrenado como base, que puede ser reutilizado para nuestras propias tareas.

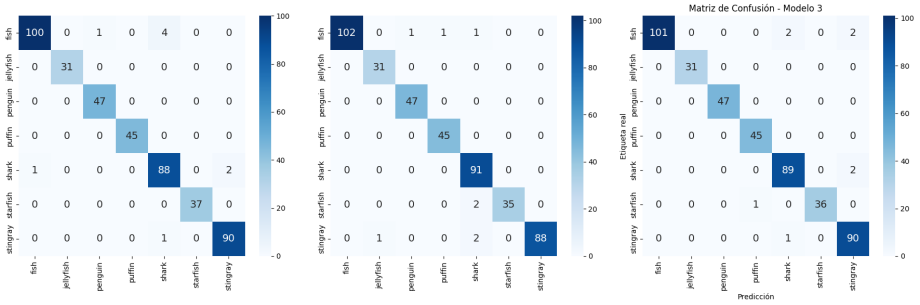


Figura 6: Matrices de confusión entrenamiento.

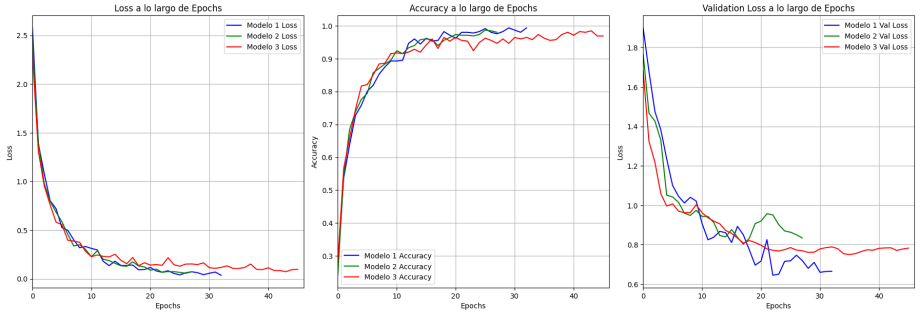


Figura 7: Matrices de confusión test + train_loss, accuracy y val_loss

# Referencias

[1] Francois Chollet y François Chollet. *Deep learning with Python*. Simon y Schuster, 2021.