universidad de león

Máster en Robótica y Sistemas inteligentes

CLASIFICACIÓN CON CNN

Aitor García Blanco

Índice

1.Introducción

En este proyecto, se tiene como objeto diseñar e implementar diferentes modelos de CNN capaces de clasificar imágenes en cinco categorías específicas.

Para lograr este objetivo, se emplean diferentes enfoques con redes neuronales convolucionales. Por un lado, tenemos la red sin pesos entrenados que entrenaremos y ajustaremos nosotros y, por otro, aplicamos *transfer learning*.

Además, intentaremos obtener los mejores resultados empleando las configuraciones más efectivas y robustas mediante el análisis de diferentes métricas.

En el siguiente repositorio se encuentra el proyecto en Python para este proyecto:

ULE-Informatica-2024-2025/vico24-AigarciabFabero: vico24-AigarciabFabero created by GitHub Classroom

En esta memoria, se representan los resultados, se comentan y se proporciona una visión global del proyecto. Para un mayor detalle, se recomienda revisar el repositorio donde se detallan los pasos a seguir.

2.Descripción del dataset

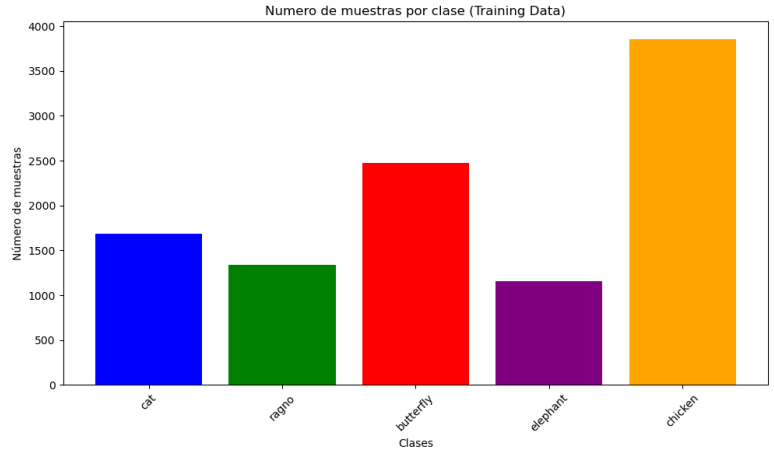
El *dataset* que se emplea para la clasificación de imágenes de animales, se puede encontrar en el siguiente repositorio: Animals-10.

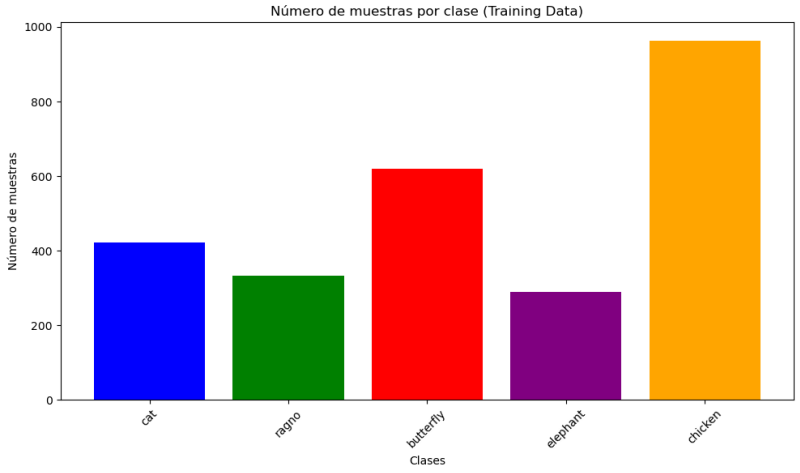
El conjunto de imágenes que nos proporciona el *dataset*, consta de 26.0000 imágenes de animales agrupados en 10 categorías de animales: perro, gato, caballo, mariposa, araña, pollo, oveja, vaca, ardilla y elefante. Para cada clase, disponemos de un número heterogéneo de imágenes con formato .jpeg y en escala de color; de dimensiones variadas. Durante todo el procesamiento, las imágenes se redimensionan con un tamaño de 244x244.

El directorio principal está dividido en carpetas, una para cada categoría. El recuento de imágenes para cada categoría varía de 2.000 a 5.000 unidades.

Tan solo podemos emplear 5 de las 10 clases por lo que nos quedamos con las siguientes: *cat, ragno, butterfly, elephant, chicken.*

Una vez fragmentamos nuestro *dataset* en conjunto de entrenamiento (80%) y test (20%), cabe destacar que sobre el conjunto de entrenamiento aplicamos *data augmentation* (*horizontal_flip=True*) con la finalidad de obtener unos mejores resultados.





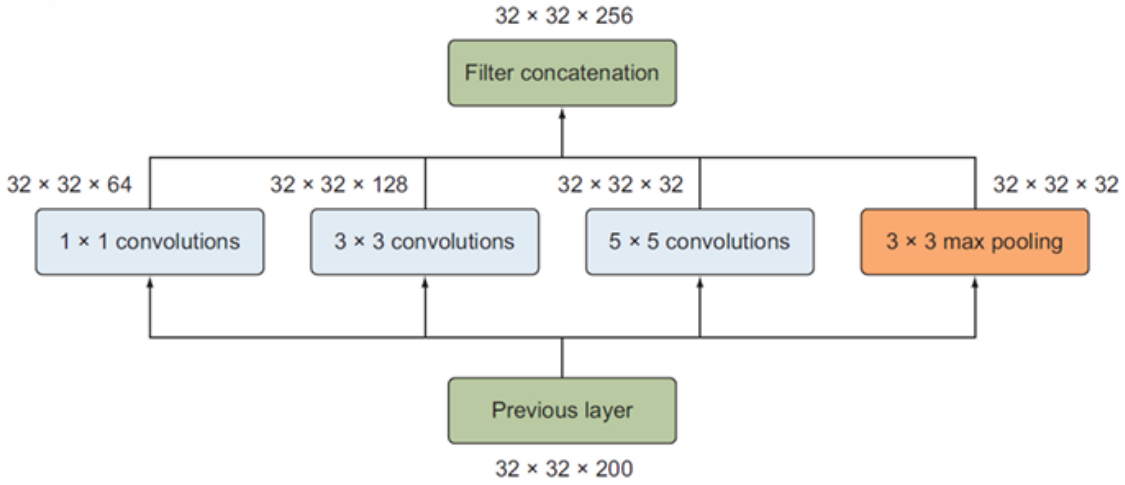
3.CNN

En este bloque se nos pide que diseñemos una red neuronal convolucional o que empleemos una arquitectura conocida (entrenada desde cero) para clasificar las imágenes. La segunda opción nos parece más interesante para poder comparar el mismo modelo tras aplicar *Transfer Learning;* como veremos posteriormente.

3.1. Fundamento teórico

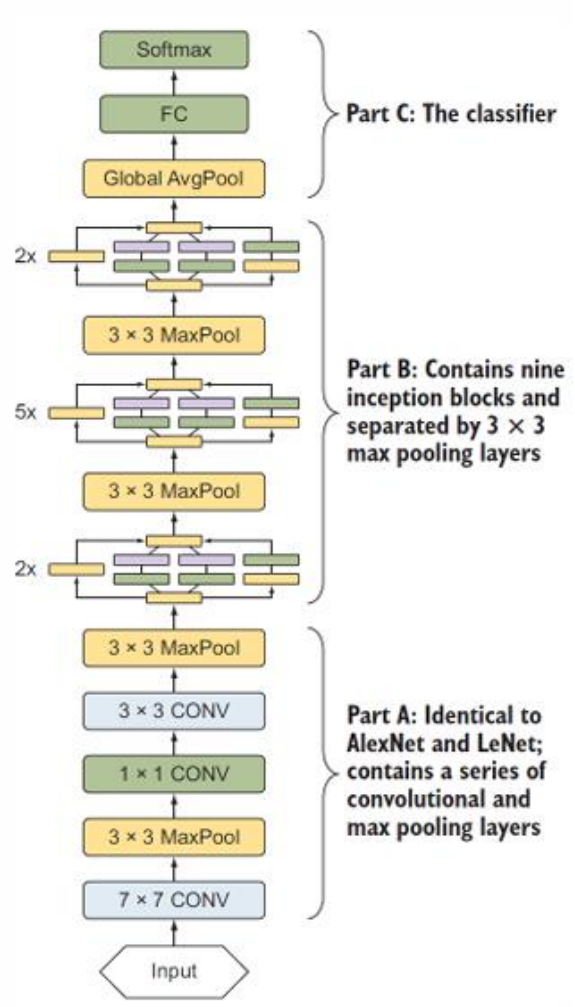
La red *Inception*, presentada en 2014 por investigadores de Google, revolucionó el aprendizaje profundo al combinar gran profundidad (22 capas) con una reducción drástica de parámetros (de 138 millones a 13 millones). Inspirada en *AlexNet* y *VGGNet*, introdujo el innovador módulo *Inception*, que permite capturar características a múltiples escalas de manera eficiente, logrando mayor precisión con menor costo computacional.

Los módulos *Inception* son el núcleo de la arquitectura y están diseñados para extraer características de múltiples escalas de manera eficiente. Estos núcleos procesan la información empleando: convoluciones 1x1 (capturan relaciones internas y reducen la dimensionalidad), convoluciones 3x3 (capturan patrones locales y más complejos) y una capa *pooling* (*max* o *average pooling*) para reducir el tamaño de las características. Los resultados de las operaciones anteriores se concatenan a lo largo de los canales.



La arquitectura de *Inception* se puede desglosar en 3 capas:

a) Idéntica a las capas de *AlexNet* y *LeNet*. Contiene un conjunto de capas convolucionales y de agrupación (*pooling*)
b) Contiene nueve módulos Inception apilados.
c) *Fully conected* y una función de activación *softwax* para realizar la multi clasificación.



3.2. Configuración

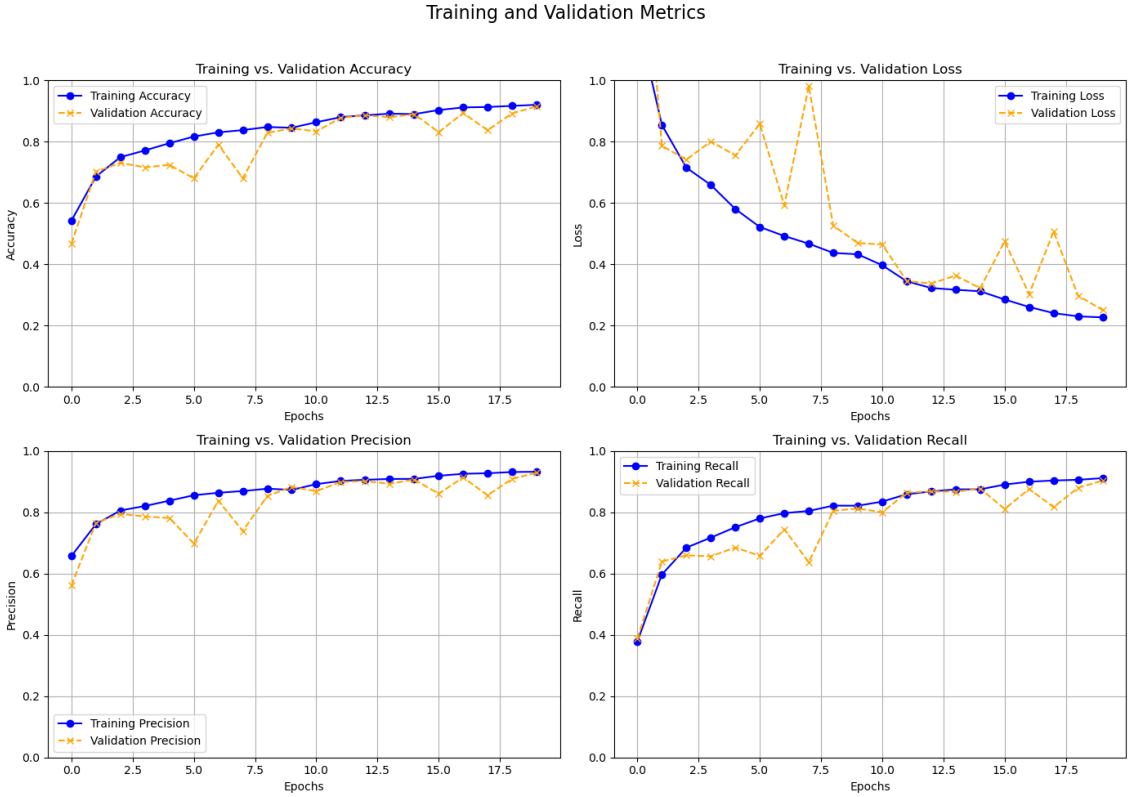
Para el correcto desarrollo de este bloque, hemos empleado el modelo InceptionV3 sin pesos, para esto, hemos establecido en nulo el parámetro *weight*. Para más detalle a cerca de la estructura de este modelo, y sobre este apartado de configuración, se recomienda tener como información complementaria la proporcionada en el repositorio del proyecto.

Una vez tenemos nuestro modelo importado, se establecen dos configuraciones para los optimizadores que se van a encargar de minimizar la función de pérdida durante el entrenamiento. Hemos empleado una configuración para el modelo Adam y otra para SGD.

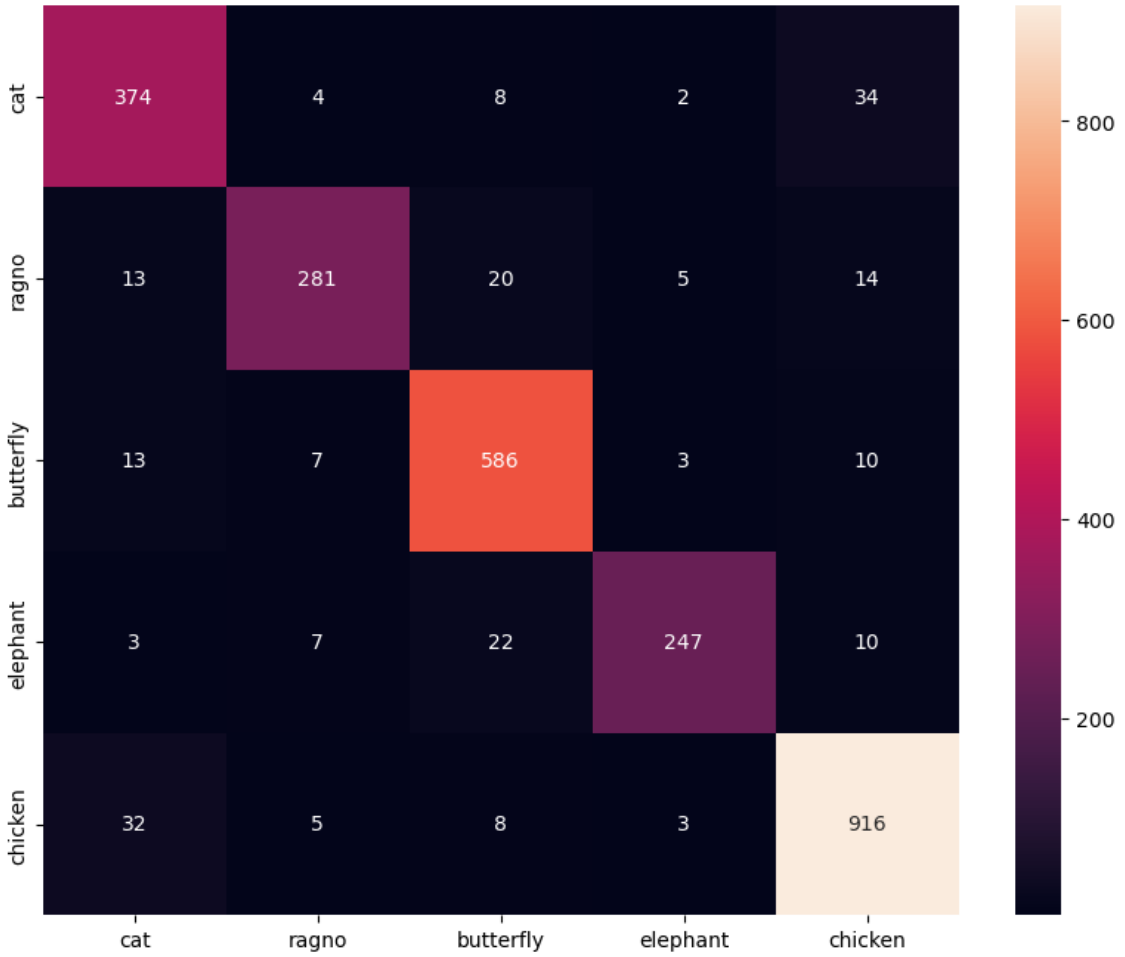
Respecto a la compilación del modelo, podemos decir que hemos empleado la función de pérdida la *categorical crossentropy*, finalmente el optimizador Adam con un *learning rate* de 0.001 y las métricas: *accuracy*, *precision* y *recall*.

Finalmente, entrenamos nuestro modelo con 20 *epoch* y un tamaño del *batch* de 15.

3.3. Resultados



	precision	recall	f1-score	support
cat	0.86	0.89	0.87	422
ragno	0.92	0.84	0.88	333
butterfly	0.91	0.95	0.93	619
elephant	0.95	0.85	0.90	289
chicken	0.93	0.95	0.94	964
accuracy			0.92	2627
macro avg	0.91	0.90	0.90	2627
weighted avg	0.92	0.92	0.91	2627



Las matrices de confusión que se muestran durante toda la memoria son sobre el conjunto de datos *test*.

En líneas generales, los resultados son muy satisfactorios. En la primera imagen, podemos ver como las diferentes métricas empleadas arrojan resultados muy positivos, sin *underfitting* y *overffiting*, se ve que el modelo generaliza bastante bien. Algunos picos puntuales de ruido, principalmente en las primeras fases de entrenamiento. El problema de esta metodología reside en los tiempos de entrenamiento y el numero de veces que repetimos los procesos de entrenamientos hasta alcanzar estos resultados, pues buscamos en todo momento, una configuración optima de parámetros.

4.Transfer Learning

Para hacer *Transfer Learning* y clasificar vamos a seleccionar la red neuronal convolucional *Inception*.

4.1. Fundamento teórico

Transfer Learning es una técnica clave en el aprendizaje profundo, que permite reutilizar modelos preentrenados para nuevas tareas, ahorrando tiempo y esfuerzo en la recolección y etiquetado de grandes cantidades de datos. En las Redes Neuronales Convolucionales (CNN), se pueden usar los modelos preentrenados como extractores de características para nuevos conjuntos de datos, aprovechando las representaciones generales que ya han aprendido.

El proceso consiste en descargar un modelo preentrenado, eliminar la parte del clasificador, añadir un nuevo clasificador adaptado al problema y reentrenar la red. Esto es particularmente útil cuando se cuenta con conjuntos de datos pequeños, ya que reduce el tiempo de entrenamiento y mejora la precisión.

Para ajustar los pesos del (*fine-tuning*), se utiliza una tasa de aprendizaje más baja, ya que estos pesos ya son bastante buenos y no deben modificarse demasiado con rapidez, especialmente mientras el nuevo clasificador se entrena desde cero.

La efectividad del aprendizaje por transferencia depende de dos factores clave:

- El tamaño del conjunto de datos objetivo.
- La similitud entre los dominios de los conjuntos de datos fuente y objetivo.

Esta técnica permite reutilizar los pesos de modelos preentrenados en conjuntos de datos estándar como *ImageNet*, simplificando el desarrollo de nuevos sistemas de visión por computadora.

Tal y como hemos visto en los apuntes de teoría, tenemos cuatro escenarios diferentes:

Escenario 1: El conjunto de datos objetivo es pequeño y similar al conjunto de datos fuente
Dado que el conjunto de datos original es similar al nuevo, es probable que las características de nivel alto de la *ConvNet* preentrenada sean relevantes para nuestro conjunto de datos. En este caso, lo mejor sería congelar la parte de extracción de características de la red y solo reentrenar el clasificador. Además, debido al tamaño reducido del nuevo conjunto de datos, afinar las capas de extracción de características podría provocar un sobreajuste a los datos.

Escenario 2: El conjunto de datos objetivo es grande y similar al conjunto de datos fuente
Como ambos dominios son similares, se puede congelar la parte de extracción de características y reentrenar el clasificador, como en el escenario 1. Sin embargo, dado que tenemos más datos en el nuevo dominio, podemos mejorar el rendimiento afinando

toda o parte de la red preentrenada con mayor confianza de que no habrá sobreajuste. Afinar toda la red no suele ser necesario, ya que las características de nivel alto son relevantes.

Escenario 3: El conjunto de datos objetivo es pequeño y diferente del conjunto de datos fuente
Dado que los conjuntos de datos son diferentes, congelar las características de nivel alto de la red preentrenada puede no ser la mejor opción, ya que estas contienen características más específicas del conjunto de datos original. En su lugar, sería mejor reentrenar capas de una parte más temprana de la red o incluso no congelar ninguna capa y afinar toda la red. Sin embargo, debido al tamaño reducido del conjunto de datos, afinar toda la red podría no ser recomendable, ya que puede hacer que el modelo se sobreajuste.

Escenario 4: El conjunto de datos objetivo es grande y diferente del conjunto de datos fuente
Dado el tamaño grande del nuevo conjunto de datos, podría parecer lógico entrenar toda la red desde cero sin usar aprendizaje por transferencia. Sin embargo, en la práctica, suele ser beneficioso inicializar los pesos desde un modelo preentrenado, ya que esto acelera la convergencia del modelo. En este caso, el tamaño del conjunto de datos proporciona confianza para afinar toda la red sin preocuparse por el sobreajuste.

4.2. Configuración

Para este procedimiento, se ha tenido en cuenta que las 5 clases de nuestro *dataset* son clases preentrenadas en el conjunto de datos de ImageNet, además, tenemos relativamente, un amplio numero de imágenes; por tanto, podemos estar tanto en el primer como en el 2 escenario.

Atendiendo a lo anterior, decidimos congelar todas las capas del modelo base y añadimos y entrenamos un nuevo clasificador con dos capas densas. Por tanto, no estamos ajustando los pesos de las capas preentrenadas, sino que las estamos empleando como un extractor de características.

Por tanto, la configuración del modelo es la siguiente. Por un lado, congelamos las capas de nuestro modelo base y le añadimos una capa *averagepooling,* una capa densa con 512 neuronas y función de activación *relu* y, una capa densa de 5 neuronas con función de activación *softmax* para poder realizar la clasificación multiclase.

A la hora del entrenamiento, empleamos el optimizador Adam con un *learning rate* de 0.001, una función de pérdida *categorical crossentropy* y las mismas métricas que en el apartado anterior. Además, se establecen tres *callback* para monitorizar el entrenamiento:

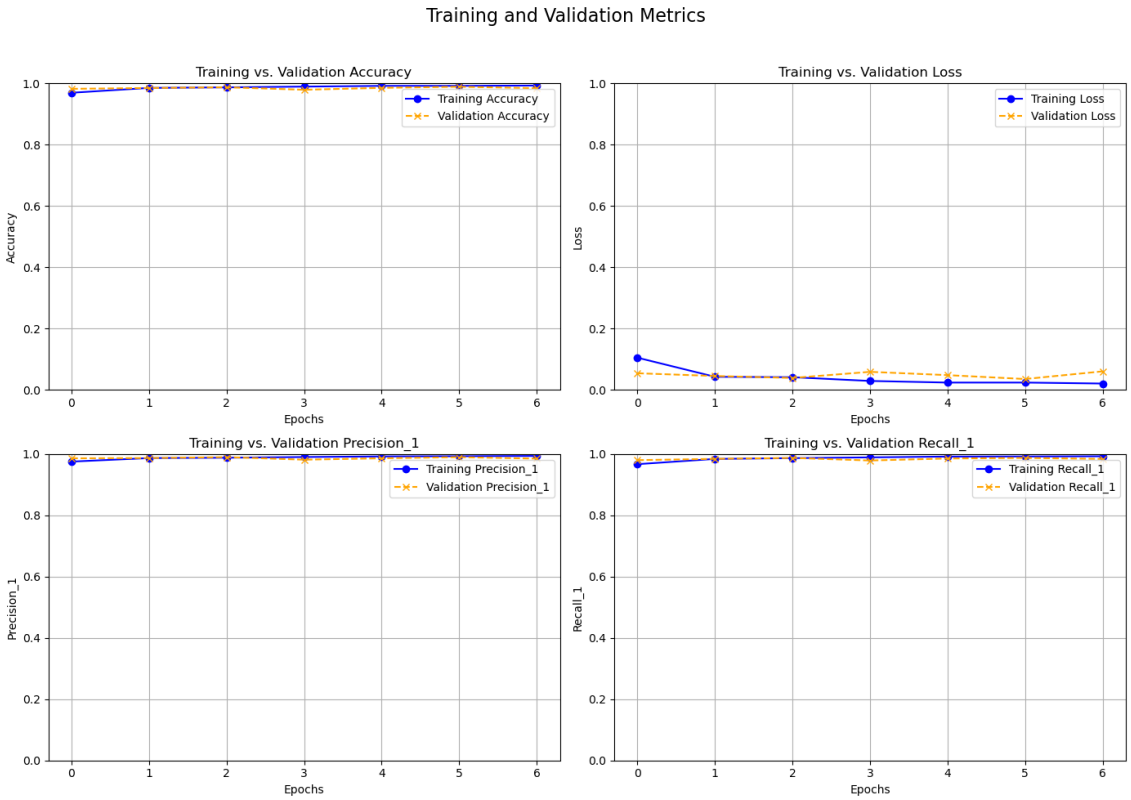
- *EarlyStopping*: Detiene el entrenamiento si la pérdida de validación no mejora después de 4 épocas.

- *ModelCheckpoint*: Guarda el mejor modelo basado en la pérdida de validación.

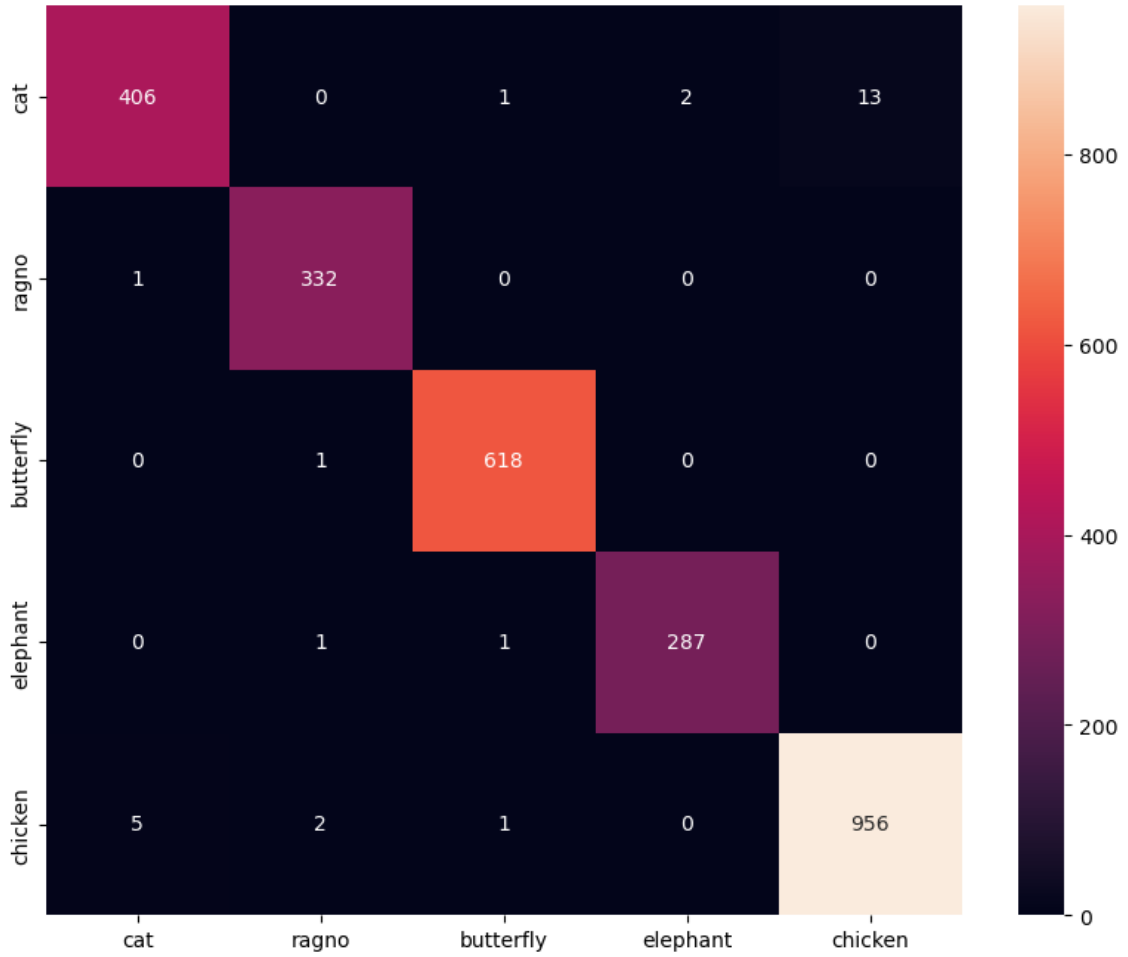
- *ReduceLROnPlateau*: Reduce la tasa de aprendizaje si la pérdida de validación no mejora después de 5 épocas.

Finalmente, entrenamos nuestro modelo con 7 *epoch* y un tamaño de *batch* de 64.

4.3. Resultados



	precision	recall	f1-score	support
cat	0.99	0.96	0.97	422
ragno	0.99	1.00	0.99	333
butterfly	1.00	1.00	1.00	619
elephant	0.99	0.99	0.99	289
chicken	0.99	0.99	0.99	964
accuracy			0.99	2627
macro avg	0.99	0.99	0.99	2627
weighted avg	0.99	0.99	0.99	2627



Los resultados son más que satisfactorios, tal y como podemos ver en las métricas de la primera y segunda imagen. Además, lo amparamos con la representación de la matriz de confusión sobre el conjunto de test. Nuestro modelo generaliza correctamente y se ajusta a los datos adecuadamente. Podemos concluir que tenemos un muy buen clasificador.

Los tiempos de entrenamiento para esta metodología son significativamente inferiores a los de la metodología anterior, tal y como podemos ver en los resultados más extendidos y recogidos en el repositorio.

5.Conclusión

En este proyecto se ha abordado un problema de clasificación multiclase aplicando dos metodologías diferentes. Por un lado, hemos empleado el modelo InceptionV3 entrenado desde cero y por otro, aprendizaje por transferencia (*transfer learning*).

Los resultados de ambas metodologías muestran un alto rendimiento de los modelos, con métricas consistentes y una matriz de confusión que respalda la precisión de los clasificadores. El empleo de técnicas como *data augmentation* nos han asegurado un entrenamiento robusto.

A pesar de que ambas metodologías arrojan resultados muy buenos, los datos y resultados obtenidos demuestran que el *transfer learning* es claramente la mejor opción, destacándose como una herramienta eficiente y efectiva para abordar la casuística específica de este proyecto.

Esta metodología, emplea el modelo preentrenado de Inception, aprovechando así, las características previamente aprendidas, ahorrando tiempo y recursos en el entrenamiento. Este enfoque ha resultado ser especialmente útil, considerando que el dataset objetivo tiene similitudes con el conjunto de datos fuente (*ImageNet*). Para conseguir igualar estos resultados con la primera metodología, sería necesario disponer de muchos recursos computacionales, tiempo y densidad de imágenes.