

Escuela Técnica Superior de Ingeniería Universidad de Huelva

Grado de Ingeniería Informática

Trabajo Fin de Grado

Aplicaciones de estrategias de Deep Learning para la
detección de animales en imágenes de fototrampeo

Carlos García Silva

Mayo 2025

Aplicaciones de estrategias de Deep Learning para la detección de animales en imágenes de fototrampeo

Resumen

La importancia de un futuro sostenible se refleja cada día con mayor claridad en nuestra sociedad. Es por ello por lo que los estudios de biodiversidad han cobrado un papel fundamental en la compresión, conservación y valoración de la vida en la Tierra.

Una de las técnicas ampliamente utilizada para la realización de dichos estudios es la técnica del fototrampeo. Se basa en el despliegue de cámaras automáticas en el entorno a estudiar para poder capturar imágenes de animales en su hábitat natural. Uno de sus principales problemas es la gran facilidad para generar grandes volúmenes de imágenes, en las que realizar un análisis manual puede resultar costoso y laborioso.

Este Trabajo de Fin de Grado presenta una red neuronal convolucional capaz de clasificar imágenes de foto-trampeo para identificar aquellas donde existe presencia de animales. El sistema propuesto ha sido desarrollado mediante el uso de la técnica de transfer learning, empleando un modelo previamente entrenado con los pesos de ImageNet, basado en la arquitectura EfficientNet-B5.

El experimento se realizó utilizando una colección de 31.670 imágenes obtenidas de cámaras de foto-trampeo, de las cuales 21.670 presentaban evidencia de presencia animal. El proceso de entrenamiento, validación y evaluación de la red se llevó a cabo utilizando una proporción del 70%, 15% y 15% del conjunto de imágenes, respectivamente.

Los resultados obtenidos muestran que el modelo propuesto alcanza valores de rendimiento excepcionales, con una accuracy cercana al 97%, loss alrededor del 1%, precision del 97%, recall del 99%, specificity del 94% y un AUC de aproximadamente 0.995.

Como conclusión, se ha logrado desarrollar una red neuronal convolucional especializada que supera el rendimiento del modelo MegaDetector. Esta mejora confirma que es posible construir sistemas más precisos adaptados a contextos concretos. Además, se sientan las bases para líneas futuras de investigación orientadas a la clasificación multicategoría o por comportamiento animal, aportando herramientas eficaces y escalables para el monitoreo ecológico automatizado.

Palabras clave: Deep Learning, redes neuronales convolucionales, transferencia aprendizaje, ImageNet, fototrampeo, clasificador, MegaDetector, fauna silvestre, EfficientNet-B5.

Applications of Deep Learning strategies for the detection of animals in cameratraps images

Abstract

The importance of a sustainable future is reflected more and more clearly in our society. This is why biodiversity studies have taken on a fundamental role in the understanding, conservation and valuation of life on Earth.

One of the techniques widely used to carry out such studies is the photo-trapping technique. It is based on the deployment of automatic cameras in the environment to be studied in order to capture images of animals in their natural habitat. One of its main problems is that it is very easy to generate large volumes of images, where manual analysis can be costly and time-consuming.

This Final Degree Project presents a convolutional neural network capable of classifying photo-trapping images to identify those where animals are present. The proposed system has been developed using the transfer learning technique, employing a model previously trained with ImageNet weights, based on the EfficientNet-B5 architecture.

The experiment was conducted using a collection of 31.670 images obtained from photo-trapping cameras, of which 21.670 showed evidence of animal presence. The network training, validation and evaluation process was carried out using a proportion of 70%, 15% and 15% of the image set, respectively.

The results obtained show that the proposed model achieves exceptional performance values, with an accuracy close to 97%, loss around 1%, precision of 97%, recall of 99%, specificity of 94% and an AUC of approximately 0.995.

In conclusion, a specialized convolutional neural network has been developed that outperforms the MegaDetector model. This improvement confirms that it is possible to build more accurate systems tailored to specific contexts. Moreover, it lays the foundations for future lines of research aimed at multi-category or animal behavior classification, providing efficient and scalable tools for automated ecological monitoring.

Keywords: Deep Learning, convolutional neural networks, transfer learning, ImageNet, photo-trapping, classifier, MegaDetector, wildlife, EfficientNet-B5.

Índice general

<i>Resumen</i>	<i>II</i>
<i>Abstract</i>	<i>IV</i>
<i>Índice de figuras</i>	<i>X</i>
<i>Índice de tablas</i>	<i>XII</i>
Capítulo 1 Propuesta de Proyecto	1
1.1 Motivación	1
1.2 Objetivos	2
1.3 Competencias	3
1.4 Hardware y Software	3
1.5 Organización de la memoria	4
Capítulo 2 Introducción y estado del arte	7
Capítulo 3 Materiales	9
3.1 Base de datos.....	9
3.2 Conjunto de datos	12
3.3 Métricas de evaluación.....	13
Capítulo 4 Metodología	15
4.1 Arquitectura de la red	15
4.2 Hiperparámetros	16
4.2.1 Batch size	16
4.2.2 Algoritmo de optimización	17
4.2.3 Función de pérdida.....	17
4.2.4 Valores numéricos de los hiperparámetros	17
4.2.5 Data augmentation.....	18
4.2.6 Early Stopping	18
4.2.7 Fine Tuning.....	18
4.3 Fase de entrenamiento	19
4.3.1 Métricas de referencia de MegaDetector	20
A.1.1 Entrenamiento de MegaClassifier	22
Capítulo 5 Resultados y discusión	40
5.1 Resultados del conjunto de prueba	40
5.2 Análisis y discusión	44
Capítulo 6 Conclusiones y trabajos futuros	46
6.1 Conclusiones técnicas	46

6.2 Trabajos futuros	47
6.3 Valoración personal	47
Referencias.....	50
Bibliografía.....	54
Anexo teórico A Introducción e historia del Deep Learning.....	58
A.2 Introducción al Deep Learning.....	58
A.3 Historia del Deep Learning	63
A.3.1 Etapa Cibernética (1940-1960).....	64
A.3.2 Etapa Conexiónismo (1980 – 1995)	66
A.3.3 Etapa Deep Learning (2006 – Actualidad).....	67
Anexo teórico B Redes neuronales	70
B.1 Fundamentos	70
B.1.1 Perceptrón	70
B.1.2 Funciones de activación	72
B.1.3 Tipos de capas	74
B.1.4 Aplicación a problemas de clasificación	75
B.1.5 Aplicación a imágenes.....	76
B.2 Desarrollo de redes neuronales.....	77
B.2.1 Métricas de evaluación.....	79
B.2.2 Estimación del error	84
B.2.3 Inicialización de los pesos	85
B.2.4 Algoritmos de optimización	86
B.2.5 Backpropagation	89
B.3 Hiperparámetros	91
B.3.1 Arquitectura de red.....	91
B.3.2 Técnica de inicialización de los pesos	92
B.3.3 Número de épocas.....	92
B.3.4 Batch Size	93
B.3.5 Almacenamiento de los pesos	93
B.3.6 Algoritmo de optimización	93
B.3.7 Métricas	95
B.3.8 Función de pérdida.....	95
B.4 Control y seguimiento	96
Anexo teórico C Redes neuronales convolucionales	102
C.1 Introducción	102
C.2 Tipos de capas	104
C.2.1 Capas de convolución.....	104
C.2.2 Capas de activación.....	107
C.2.3 Capas de pooling	107
C.2.4 Capas upsampling y transposed convolution	108

C.2.5 Capas de softmax	112
C.2.6 Capas fully connected	112
C.3 Arquitecturas populares	112
C.3.1 AlexNet	112
C.3.2 VGG	113
C.3.3 GoogleNet.....	113
C.3.4 ResNet	114
C.3.5 DenseNet	115
C.3.6 Arquitectura utilizada en este trabajo	116
Anexo teórico D <i>MegaDetector</i>.....	120

Índice de figuras

Figura 1: Distribución de las clases en el dataset original	9
Figura 2: Distribución binaria del dataset original.....	10
Figura 3: Ejemplo de imagen diurna con animal camuflado resaltado	10
Figura 4: Ejemplo de imagen diurna con presencia animal	10
Figura 5: Ejemplo de imagen nocturna con animal camuflado resaltado	11
Figura 6: Ejemplo de imagen nocturna con presencia animal	11
Figura 7: Ejemplo de imagen diurna con presencia animal en un encuadre muy abierto.....	11
Figura 8: Distribución binaria del dataset final	12
Figura 9: Esquema del modelo principal MegaClassifier_C.....	16
Figura 10: Matriz de confusión ideal de train	21
Figura 11: Matriz de confusión ideal de validation	21
Figura 12: Matriz de confusión de validation de MegaDetector umbral=0.1.....	21
Figura 13: Matriz de confusión de train de MegaDetector umbral=0.1.....	21
Figura 14: Gráficas de entrenamiento de la versión 1	23
Figura 15: Gráficas de entrenamiento de la versión 2	24
Figura 16: Gráficas de entrenamiento de la versión 3	26
Figura 17: Gráfica teórica de LRFinder.....	27
Figura 18: Gráfica del método LRFinder aplicada en MegaClassifier_C	28
Figura 19: Gráficas de entrenamiento de la versión 4	29
Figura 20: Comparativa de matriz de confusión de MegaDetector con umbral por defecto y MegaClassifier_C en su versión v4.2	30
Figura 21: Gráficas de entrenamiento de la versión 5	32
Figura 22: Gráficas de entrenamiento de la versión 6	33
Figura 23: Gráficas de entrenamiento de la versión 7	34
Figura 24: Gráficas de entrenamiento de la versión 8	35
Figura 25: Comparativa de matrices de confusión de validación de MegaDetector y MegaClassifier_C con umbrales por defecto.....	36
Figura 26: Comparativa de matrices de confusión de validación de MegaDetector y MegaClassifier_C con umbrales típicos.....	37
Figura 27: Comparativa de matrices de confusión de validación de MegaDetector y MegaClassifier_C con umbrales conservadores.....	39
Figura 28: Comparativa de matrices de confusión de prueba de MegaDetector y MegaClassifier_C con umbrales conservadores.....	41
Figura 29: Comparativa de matrices de confusión de prueba de MegaDetector y MegaClassifier_C con umbrales por defecto.....	42
Figura 30: Comparativa de matrices de confusión de prueba de MegaDetector y MegaClassifier_C con umbrales típicos.....	43
Figura 31: Curva ROC y matriz de confusión del modelo final sobre el conjunto de prueba.....	44
Figura 32: Etapas del proceso de Machine Learning.....	59
Figura 33: Esquema de ejemplos de aplicaciones del Machine Learning.....	61
Figura 34: Comparativa entre Machine Learning y Deep Learning.....	63
Figura 35: Frecuencia de aparición de conceptos relacionados con el Machine Learning en publicaciones a lo largo de la historia	64
Figura 36: Demostración de cómo la función lógica XOR no puede modelarse linealmente.....	65
Figura 37: Representación del Perceptrón multicapa y el algoritmo Backpropagation	66
Figura 38: Arquitectura LeNet	67

<i>Figura 39: Arquitectura AlexNet.....</i>	68
<i>Figura 40: Anatomía de una neurona biológica</i>	71
<i>Figura 41: Perceptrón y su proceso de aprendizaje.....</i>	72
<i>Figura 42: Estructura básica de una red neuronal multicapa.....</i>	74
<i>Figura 43: Transformación softmax y codificación one-hot</i>	76
<i>Figura 44: Ejemplo de flattening.....</i>	77
<i>Figura 45: Etapas de entrenamiento e inferencia de un modelo de Machine Learning</i>	78
<i>Figura 46: Ejemplo de matriz de confusión para 4 clases.....</i>	81
<i>Figura 47: Matriz de confusión considerando únicamente las clases positivas y negativas</i>	81
<i>Figura 48: Ejemplo de gráfica de Curva ROC.....</i>	83
<i>Figura 49: Representación gráfica de la dirección del gradiente</i>	87
<i>Figura 50: Comparativa de underfitting y overfitting</i>	93
<i>Figura 51: Diferencia gráfica entre mínimo aplanado y mínimo profundo.....</i>	94
<i>Figura 52: Efecto de diferentes tasas de aprendizaje en un entreno.....</i>	94
<i>Figura 53: Ejemplo de gráficas de loss y accuracy en un entreno de 100 epochas</i>	96
<i>Figura 54: Áreas del lóbulo cerebral posterior</i>	102
<i>Figura 55: Ejemplo de extracción de características en capas de convolución.....</i>	103
<i>Figura 56: Detección de bordes sobre una imagen mediante convolución.....</i>	104
<i>Figura 57: Operación de convolución</i>	105
<i>Figura 58: Operaciones de maxpooling y averagepooling.....</i>	108
<i>Figura 59: Técnica de Nearest Neighbor</i>	108
<i>Figura 60: Técnica de Bed of Nails.....</i>	109
<i>Figura 61: Técnica de maxpooling y maxunpooling</i>	109
<i>Figura 62: Operación de convolución como multiplicación de matrices</i>	110
<i>Figura 63: Operación de convolución traspuesta</i>	111
<i>Figura 64: Arquitetura VGG16</i>	113
<i>Figura 65: Arquitectura GoogleNet</i>	114
<i>Figura 66: Arquitectura ResNet-50</i>	115
<i>Figura 67: Arquitectura DenseNet</i>	115
<i>Figura 68: Tipos de modulos de EfficientNet</i>	118
<i>Figura 69: Arquitectura EfficientNet-B5</i>	118
<i>Figura 70: Arquitectura YOLOv5</i>	120

Índice de tablas

<i>Tabla 1: Número de imágenes de cada clase en cada división del dataset.....</i>	13
<i>Tabla 2: Cobertura de positivos de MegaDetector con diferentes umbrales</i>	20
<i>Tabla 3: Métricas de MegaDetector sobre validation</i>	22
<i>Tabla 4: Comparativa MegaDetector y MegaClassifier_C sobre el conjunto de validación</i>	30
<i>Tabla 5: Comparativa MegaDetector con MegaClassifier_C con umbrales por defecto</i>	36
<i>Tabla 6: Comparativa MegaDetector con MegaClassifier_C con umbrales típicos.....</i>	37
<i>Tabla 7: Comparativa MegaDetector con MegaClassifier_C con umbrales conservadores</i>	38
<i>Tabla 8: Resumen de métricas MegaClassifier y MegaDetector sobre conjunto de validación</i>	39
<i>Tabla 9: Umbrales de los modelos</i>	40
<i>Tabla 10: Resultados del conjunto de prueba usando umbral conservador en MegaDetector y MegaClassifier_C</i>	41
<i>Tabla 11: Resultados del conjunto de prueba usando umbral por defecto en MegaDetector y MegaClassifier_C</i>	42
<i>Tabla 12: Resultados del conjunto de prueba usando umbral típico en MegaDetector y MegaClassifier_C</i>	43
<i>Tabla 13: Métricas del modelo final.....</i>	44
<i>Tabla 14: Principales funciones de activación.....</i>	73
<i>Tabla 15: Tabla comparativa de las principales arquitecturas disponibles en Tensorflow.....</i>	117

Capítulo 1 Propuesta de Proyecto

En el presente capítulo introductorio, se exponen las motivaciones que han fundamentado la realización de este trabajo, los objetivos que se han establecido y el sistema implementado para su consecución. Asimismo, se detalla el desarrollo de dichos objetivos y las diversas tecnologías empleadas.

1.1 Motivación

En años recientes, se han observado significativos avances en el ámbito de la tecnología, lo cual ha generado un impacto sustancial en la sociedad contemporánea. En cierta medida, se puede afirmar que la dependencia hacia esta área es una realidad en el contexto cotidiano. Diversos analistas consideran que la actual situación representa una nueva revolución industrial. En este sentido, se puede afirmar que, del mismo modo que sucedió con la industria textil en la primera revolución industrial, la electricidad en la segunda y la electrónica en la tercera, hoy en día se puede constatar que la informática se erige como uno de los pilares que han permitido identificar a esta nueva revolución como la denominada Industria 4.0.

En el campo de la informática, se ha observado un aumento significativo en la implementación de algoritmos y técnicas de inteligencia artificial en los últimos años. Este fenómeno se atribuye al incremento en la capacidad de procesamiento y al vasto volumen de datos disponibles en la actualidad, en contraste con la limitada disponibilidad de hace unos años. En particular, nos referimos a los algoritmos de aprendizaje profundo (Deep Learning)¹, que se fundamentan principalmente en redes neuronales artificiales.

Dentro del ámbito de la inteligencia artificial, los algoritmos han experimentado una notable evolución en su capacidad de procesamiento de datos. La implementación de estos algoritmos se lleva a cabo durante el proceso de entrenamiento, mediante la utilización de vastas cantidades de datos. Además, estos algoritmos se ejecutan en sistemas con capacidades de cómputo que, hasta hace una década, se consideraban inalcanzables para las máquinas. Los resultados obtenidos recientemente han superado las expectativas, logrando niveles de rendimiento que anteriormente eran impensables para las máquinas y, en algunos casos, incluso superando a los humanos en ciertas tareas. Este avance representa una significativa apertura a la capacidad de abordar nuevos y complejos problemas que, anteriormente, solo podían resolverse mediante el ingenio humano.

¹ Debido a la popularidad del término Deep Learning durante el resto del trabajo lo utilizaremos en sustitución del término aprendizaje profundo.

El propósito de este análisis es desarrollar e implementar una red neuronal convolucional (CNN)² con la capacidad de clasificar imágenes obtenidas mediante fototrampeo, diferenciando entre aquellas que contienen presencia de animales y las que no. Este sistema de clasificación se aplicará sobre conjuntos de datos reales con el fin de explorar su precisión y eficacia en el contexto del monitoreo de la fauna silvestre.

Para evaluar el desempeño del modelo propuesto, se compararán los resultados obtenidos con los de MegaDetector³, un modelo de detección de fauna ampliamente utilizada y desarrollada por Microsoft. A pesar de que MegaDetector está orientado a la detección, sus salidas pueden reinterpretarse como una forma de clasificación binaria (presencia animal/vacía), lo que lo convierte en una referencia adecuada para establecer una comparación de rendimiento. Esta comparación permitirá analizar las ventajas y limitaciones dentro de un marco de conservación y automatización del análisis de imágenes de fototrampeo.

Este proyecto constituye una oportunidad para profundizar en el conocimiento y la aplicación de técnicas avanzadas de Deep Learning, que no se abordan en el plan de estudios del Grado en Ingeniería Informática. Además, posee una clara relevancia práctica en el ámbito de la conservación de la biodiversidad. La implementación de CNN en la clasificación de imágenes de fototrampeo permite optimizar el monitoreo de la fauna, facilitando el análisis de datos con mayor eficiencia y precisión. Este avance no solo contribuye al desarrollo de nuevas metodologías en el procesamiento de imágenes, sino que también abre la posibilidad de generar herramientas de apoyo en estudios ecológicos y en la preservación de especies, lo que destaca la importancia de la inteligencia artificial en la resolución de problemas del mundo real.

1.2 Objetivos

En el marco de la presente investigación, se han planteado dos objetivos primordiales a alcanzar:

- **Adquisición de conocimiento:** Introducción y estudio teórico de las técnicas de Deep Learning, con el propósito de comprender su naturaleza, su evolución hasta la actualidad y los fundamentos necesarios para su aplicación en el problema específico de interés.
- **Implementación:** En segundo lugar, se implementará un sistema basado en CNN que permitirá llevar a cabo una clasificación eficaz de la presencia o no, de animales en imágenes de fototrampeo.

Para una mayor profundización en el tema, se propone una subdivisión de los objetivos principales en los siguientes puntos: En primer lugar, es necesario realizar una exhaustiva revisión bibliográfica y un análisis de los conceptos teóricos básicos acerca del Deep Learning.

² Siglas del término en inglés Convolutional Neural Network. Por motivos de simplicidad, durante el resto del trabajo utilizaremos dichas siglas en sustitución del término red neuronal convolucional.

³ Enlace al repositorio oficial del modelo de detección:
<https://github.com/microsoft/CameraTraps/tree/main>

En segundo lugar, se obtendrá, analizará y preparará un conjunto de imágenes⁴ para el entrenamiento, la validación y la evaluación de los modelos implementados. En tercer lugar, se procederá al diseño, implementación y evaluación de varios modelos entrenados para la detección de animales en imágenes de foto-trampeo. Por último, se llevará a cabo una búsqueda e implementación de métricas para la evaluación de los modelos, que permitirá cuantificar objetivamente la calidad de los resultados obtenidos.

1.3 Competencias

El desarrollo de este proyecto ha permitido la aplicación y el refuerzo de diversas competencias del Grado en Ingeniería Informática, particularmente aquellas vinculadas con el aprendizaje computacional y el diseño e implementación de sistemas basados en inteligencia artificial. En particular, se ha focalizado en el estudio y la aplicación de las CNN para la clasificación de imágenes de fototrampeo. Este proceso ha implicado una revisión exhaustiva del estado del arte en Deep Learning, así como la experimentación con modelos diseñados específicamente para la detección de fauna en imágenes digitales.

En contraste, la imperativa de operar con datasets de un entorno real ha posibilitado la evolución de competencias en la extracción automática de información y filtrado de imágenes para potenciar la precisión de los modelos. En este sentido, el trabajo no solo ha servido como un ejercicio de profundización en técnicas de Deep Learning que no se abordan con detalle en el plan de estudios, sino también ha permitido aplicar estos conocimientos a un caso práctico de relevancia en el ámbito de la conservación de la fauna, evidenciando la importancia del aprendizaje computacional en la solución de problemas en situaciones reales.

En resumen, las principales competencias a desarrollar son la capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.

1.4 Hardware y Software

Para poder alcanzar las metas anteriormente establecidas de manera efectiva, resulta necesario el uso de un sistema cuyo hardware posea la capacidad de ejecutar algoritmos de Deep Learning. Además, dicho sistema debe estar equipado con un software especializado en el diseño y creación de modelos que hagan uso de estos algoritmos.

En lo que respecta al software empleado, la implementación del modelo se realizará en el lenguaje de programación Python 3.8 debido a su amplia popularidad en aplicaciones de aprendizaje automático. El framework seleccionado para la ejecución de este proyecto es

⁴ Debido a su popularidad, a partir de este momento, se utilizará el término en inglés *dataset* en sustitución del término conjunto de datos/imágenes.

TensorFlow⁵, debido a su exhaustiva documentación y activa comunidad de usuarios, lo que facilitará la implementación del proyecto. Desde su versión 2.0, TensorFlow incorpora de forma nativa la librería Keras⁶, que permite la creación de redes neuronales desde un alto nivel gracias a su estructura de capas de abstracción. Esta integración proporciona una metodología eficiente para la concepción de modelos de Deep Learning, permitiendo descender en la abstracción solo cuando sea necesario y de forma más precisa.

Para la implementación y el entrenamiento del modelo de clasificación se utilizarán aplicaciones como Jupyter Notebook⁷ y Visual Studio Code⁸, junto con el entorno de desarrollo proporcionado por Anaconda⁹, que nos permitirá instalar y gestionar fácilmente las librerías necesarias, como Tensorflow, OpenCV¹⁰, Numpy¹¹ y otras más.

Para la implementación del proyecto, se ha utilizado un MacBook Pro de 14 pulgadas, equipado con el chip Apple M3, 16 GB de memoria unificada y un almacenamiento SSD de 512 GB. Este dispositivo ha permitido la ejecución eficiente de los experimentos, gracias a su CPU de 8 núcleos, optimizada para tareas de alto rendimiento y eficiencia, así como su GPU de 10 núcleos, que incorpora tecnologías avanzadas como Dynamic Caching, Mesh Shading y Ray Tracing acelerado por hardware. Además, la librería TensorFlow ha experimentado actualizaciones que han posibilitado su compatibilidad nativa con la nueva arquitectura de procesadores Apple Silicon, lo que ha conducido a un notable incremento en su rendimiento.

1.5 Organización de la memoria

Además del primer capítulo, en el que se presenta la propuesta, la motivación y los objetivos del trabajo, así como las competencias desarrolladas y los recursos hardware y software empleados, esta memoria se organiza en los siguientes capítulos:

- **Capítulo 2 Introducción y estado de arte:** Se presenta un análisis exhaustivo de la implementación de técnicas de Deep Learning en el ámbito de la clasificación de imágenes. Este capítulo examina estudios previos y tecnologías relevantes en el contexto del reconocimiento de fauna mediante foto-trampeo.
- **Capítulo 3 Materiales:** Se aborda la descripción de los materiales empleados en el desarrollo del modelo. En este capítulo se describen los datos utilizados en la implementación del modelo, incluyendo la base de datos empleada, los conjuntos de datos generados para el entrenamiento, validación y evaluación; y las métricas utilizadas para medir el desempeño del sistema.

⁵ <https://www.tensorflow.org>

⁶ <https://keras.io/>

⁷ <https://jupyter.org/>

⁸ <https://code.visualstudio.com/>

⁹ <https://www.anaconda.com/>

¹⁰ <https://opencv.org/>

¹¹ <https://numpy.org/>

- **Capítulo 4 Metodología:** Como su propio nombre indica, se centra en la metodología empleada, detallando las estrategias implementadas en el desarrollo del modelo. En este sentido, se aborda la selección de la arquitectura de la red neuronal, los hiperparámetros utilizados y el proceso de entrenamiento del modelo.
- **Capítulo 5 Resultados y discusión:** Presenta los resultados obtenidos en el conjunto de evaluación, analizando su rendimiento en términos de las métricas establecidas y discutiendo las fortalezas y limitaciones del enfoque adoptado.
- **Capítulo 6 Conclusiones y trabajos futuros:** Se presentan las conclusiones técnicas extraídas del estudio, se identifican posibles mejoras y líneas de trabajo futuro, y se incluye una valoración personal sobre la experiencia adquirida durante la realización del proyecto.

La memoria prosigue con los capítulos destinados a la bibliografía y referencias, en el cual, como su propio nombre indica se efectúan las referencias de los artículos, libros y recursos empleados para la elaboración del trabajo.

Además de los capítulos principales, la memoria contiene una serie de anexos teóricos adicionales que profundizan en los conceptos fundamentales del trabajo. Se encuentra organizado de la siguiente manera:

- **Anexo teórico A: Introducción al Deep Learning:** Se presentan los principios básicos del Deep Learning, destacando su evolución y relevancia en la actualidad.
- **Anexo teórico B: Redes Neuronales:** En este apartado se abordan los fundamentos del funcionamiento de las redes neuronales, desde su desarrollo hasta las diferentes etapas del entrenamiento e inferencia, detallando aspectos como la configuración de hiperparámetros y el control del proceso de entrenamiento.
- **Anexo teórico C: Redes Neuronales Convolucionales:** Se analiza la estructura y funcionamiento de las CNN, describiendo los tipos de capas utilizadas, las arquitecturas más populares y la arquitectura específica empleada en este trabajo.
- **Anexo teórico D: MegaDetector:** Finalmente se presenta MegaDetector, utilizado para el preprocesamiento de las imágenes de foto-trampeo utilizadas como entrada para nuestro modelo clasificador, explicando su funcionamiento y su integración dentro del sistema desarrollado.

Capítulo 2 Introducción y estado del arte

El monitoreo de la biodiversidad es esencial para la conservación de los ecosistemas y la gestión sostenible de la fauna silvestre. Las cámaras de fototrampeo se han consolidado como una herramienta clave en este ámbito, permitiendo la captura de imágenes en entornos naturales sin perturbar la actividad de las especies. Sin embargo, el análisis manual de las vastas cantidades de imágenes generadas es una tarea ardua y propensa a errores, lo que ha impulsado la búsqueda de soluciones automatizadas mediante técnicas de visión por computador y aprendizaje profundo.

Las redes neuronales convolucionales (CNN) han demostrado un rendimiento sobresaliente en tareas de clasificación de imágenes, gracias a su capacidad para aprender representaciones jerárquicas de los datos visuales. Modelos como AlexNet [1], VGGNet [2], ResNet [3], Inception [4] y EfficientNet [5] han establecido nuevos estándares en precisión y eficiencia.

No obstante, el entrenamiento de estas arquitecturas desde cero requiere conjuntos de datos extensos y recursos computacionales significativos. La transferencia de aprendizaje surge como una solución eficaz, permitiendo reutilizar modelos preentrenados en grandes bases de datos, como ImageNet, y ajustarlos a tareas específicas con conjuntos de datos más pequeños. Esta estrategia ha demostrado ser particularmente útil en aplicaciones ecológicas, donde la recopilación de datos etiquetados puede ser limitada [6].

La aplicación de CNN con transferencia de aprendizaje en el análisis de imágenes de fototrampeo ha ganado tracción en la última década. Norouzzadeh et al. [7] desarrollaron un sistema basado en CNN capaz de identificar especies animales en imágenes de cámaras trampa con una precisión comparable a la de expertos humanos. Tabak et al. [8] extendieron este enfoque para clasificar múltiples especies en diferentes ecosistemas, demostrando la escalabilidad de la metodología.

Estos estudios destacan la eficacia de las CNN en la automatización del análisis de imágenes de fototrampeo, reduciendo significativamente el tiempo y esfuerzo requeridos para procesar grandes volúmenes de datos.

A pesar de los avances, la clasificación automática de imágenes de fototrampeo enfrenta varios desafíos:

- **Variabilidad en las condiciones de captura:** Las imágenes pueden presentar diferencias significativas en iluminación, ángulo y calidad, lo que complica la generalización de los modelos.
- **Desequilibrio de clases:** Algunas especies están sobrerepresentadas en los conjuntos de datos, mientras que otras aparecen con menor frecuencia, lo que puede sesgar el entrenamiento del modelo.

- **Presencia de imágenes vacías o irrelevantes:** Las cámaras trampa a menudo capturan imágenes sin presencia de fauna, lo que introduce ruido en los datos y aumenta la carga de procesamiento.

Para mitigar estos problemas, se han propuesto técnicas como el aumento de datos, la normalización de imágenes y el uso de modelos de detección previos a la clasificación.

MegaDetector es un modelo de inteligencia artificial desarrollado por Microsoft que permite detectar automáticamente la presencia de animales, personas y vehículos en imágenes de cámaras trampa [9]. Entrenado con millones de imágenes de diversos ecosistemas, MegaDetector facilita el filtrado de imágenes vacías o irrelevantes, optimizando el flujo de trabajo en proyectos de monitoreo de fauna.

Aunque MegaDetector no identifica especies específicas, su integración como paso previo a la clasificación permite reducir significativamente el volumen de datos a procesar, enfocando los recursos computacionales en imágenes relevantes. Una descripción técnica detallada de MegaDetector se encuentra en el Anexo teórico D MegaDetector.

Este Trabajo de Fin de Grado tiene como objetivo desarrollar un sistema automatizado para la clasificación de imágenes de fototrampeo mediante CNN y transferencia de aprendizaje, utilizando pesos preentrenados en ImageNet. La propuesta incluye la integración de MegaDetector como etapa inicial para filtrar imágenes sin presencia de fauna, mejorando la eficiencia del proceso.

El sistema se entrenará y evaluará utilizando un conjunto de datos específico de imágenes de cámaras trampa, aplicando técnicas de aumento de datos y ajuste fino (fine-tuning) para optimizar el rendimiento del modelo. Se espera que esta metodología reduzca el tiempo y esfuerzo necesarios para el análisis de imágenes de fototrampeo, proporcionando una herramienta eficaz para el monitoreo de la biodiversidad.

La innovación de esta propuesta radica en la combinación de detección previa mediante MegaDetector con la clasificación basada en CNN y transferencia de aprendizaje, abordando los desafíos específicos del análisis de imágenes de fototrampeo y contribuyendo al desarrollo de soluciones eficientes en el ámbito de la conservación de la fauna silvestre.

Capítulo 3 Materiales

En el presente capítulo se aborda la descripción de los recursos empleados durante el desarrollo del trabajo, iniciando con la base de datos que sirve como punto de acceso al modelo, detallando su procedencia, clases disponibles y distribución de instancias. Posteriormente, se especifica la división de los datos en conjuntos de entrenamiento, validación y prueba, así como los criterios seguidos para dicha partición. Por último, se enumeran las métricas de evaluación empleadas para valorar el rendimiento del modelo propuesto, tanto durante la fase de entrenamiento como en los experimentos finales.

3.1 Base de datos

Para la implementación de este modelo, capaz de clasificar imágenes con o sin presencia animal, se ha proporcionado un conjunto de datos compuesto por imágenes de fototrampeo previamente clasificadas.

Sin embargo, esta categorización inicial no se alineaba con nuestro objetivo, ya que se basaba en la clasificación taxonómica de las especies presentes en las imágenes. Por consiguiente, fue necesario realizar una adaptación minuciosa para ajustarla a las características específicas de nuestro caso de estudio.

Originalmente, se nos proporcionó un conjunto de datos que contenía 19 clases taxonómicas distintas, sumando un total de 31.670 imágenes, como se muestra en la Figura 1. Sin embargo, dado que nuestro problema de clasificación se basa en una dicotomía binaria, mantendremos la clase denominada *Vacia* para indicar la ausencia de animales en la imagen.

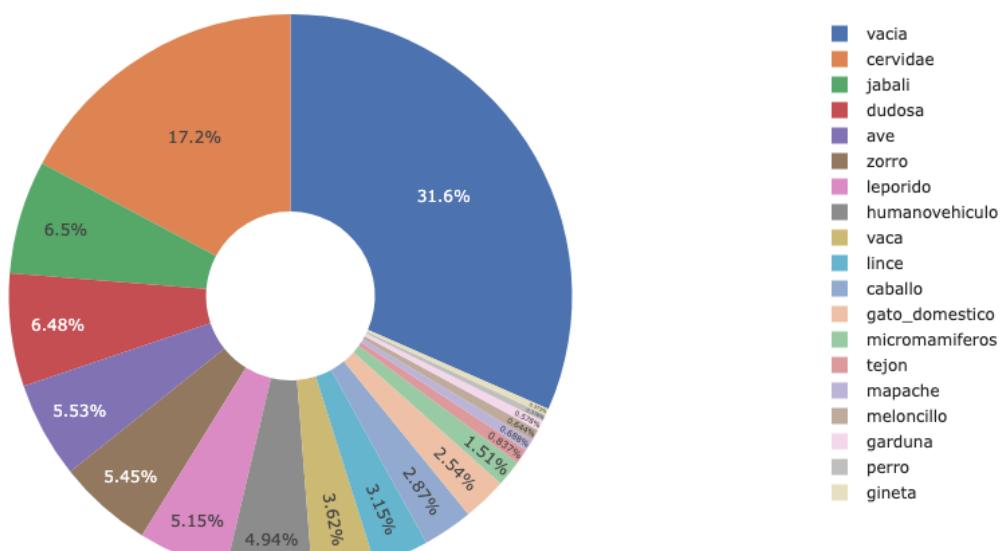


Figura 1: Distribución de las clases en el dataset original

Por otro lado, fusionaremos las demás clases, correspondientes a diferentes especies, para formar una única categoría llamada Animal. Como se evidencia en la Figura 2, se obtiene la proporción de aproximadamente 30% del conjunto de imágenes con ausencia de animales y un % con presencia. Estos datos ya nos adelantan que tendremos que trabajar con técnicas que tengan en cuenta un desbalance entre las clases.

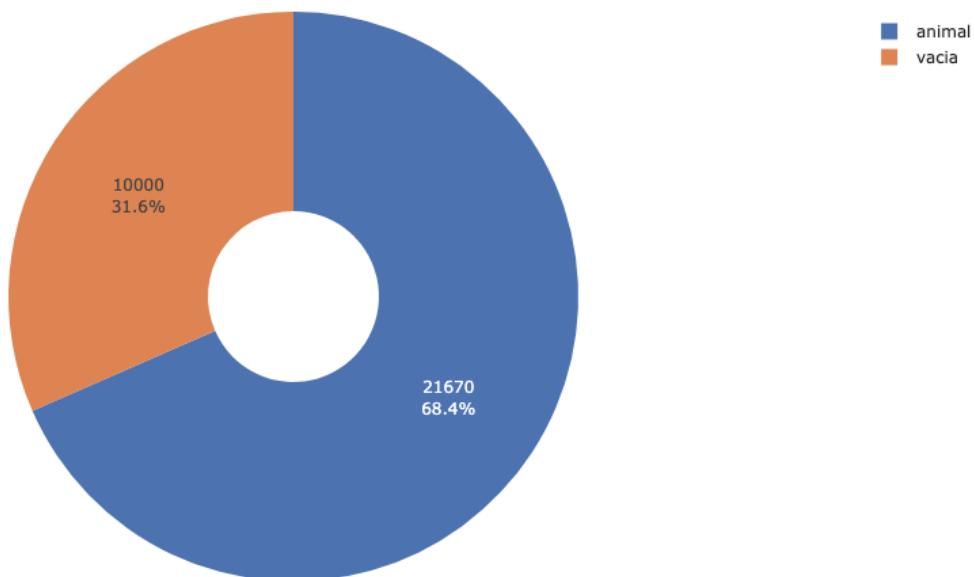


Figura 2: Distribución binaria del dataset original

Mediante el análisis de la composición de las imágenes, se evidencia la presencia de una diversidad de elementos. En primer lugar, se observa que algunas de las imágenes fueron capturadas durante el día (como la Figura 4), en las cuales el animal ocupa una porción significativa del plano. En segundo lugar, se identifica que, en aquellas imágenes capturadas durante el día, puede ser difícil discernir la ubicación del animal, ya sea debido a su camuflaje en el entorno (como se muestra en la Figura 3) o a la comparación de su tamaño con el encuadre del plano (como se observa en la Figura 7).



Figura 4: Ejemplo de imagen diurna con presencia animal



Figura 3: Ejemplo de imagen diurna con animal camuflado resaltado



Figura 7: Ejemplo de imagen diurna con presencia animal en un encuadre muy abierto

En oposición a las imágenes previamente mencionadas, se presentan aquellas que fueron capturadas durante el período nocturno, lo que implica que fueron tomadas en condiciones de baja luminosidad y, por consiguiente, con una marcada ausencia de color en la fotografía. En este tipo de imágenes se observan situaciones similares al caso anterior, donde el animal puede cubrir una porción significativa del encuadre (ver Figura 6) o también animales camuflados en el entorno nocturno (ver Figura 5).



Figura 6: Ejemplo de imagen nocturna con presencia animal



Figura 5: Ejemplo de imagen nocturna con animal camuflado resaltado

La disponibilidad de un conjunto de datos de fototrampeo que abarque una amplia gama de situaciones, composiciones, condiciones ambientales y tipos de terreno constituye una ventaja significativa en el entrenamiento de modelos de aprendizaje profundo. Además, la presencia de diferentes niveles de iluminación, ángulos de captura, fondos y especies animales permitirá al modelo aprender patrones más robustos y desarrollar una mayor capacidad para enfrentarse a casos complejos.

No obstante, durante el análisis de las diferentes casuísticas que acabamos de exponer, se observó que las imágenes pertenecientes a la clase original denominada como “dudosa”, suponían unas situaciones extremas y, por tanto, demasiado complejas para que un ser humano pudiera catalogarlas, de hecho, el propio nombre de la clase lo autodescribe. Dichos hallazgos apuntan hacia la posibilidad de que surgieran problemas relacionados con su uso, por lo que su implementación ha sido descartada.

Además de la eliminación de la clase dudosa, se observó la presencia de rutas de imágenes duplicadas, por lo que se llevó a cabo una meticulosa limpieza, teniendo como resultado una colección con un total de 28.567 imágenes, distribuidas según se muestra en la Figura 8.

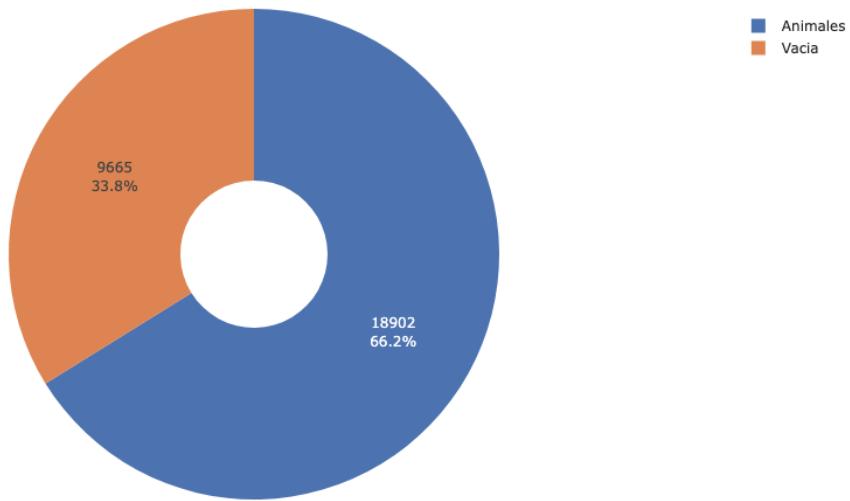


Figura 8: Distribución binaria del dataset final

3.2 Conjunto de datos

Una vez concluido el análisis de los datos obtenidos en el paso anterior, se procede a la división del conjunto de datos en tres subconjuntos. Dividir el conjunto de datos en tres subconjuntos es una práctica fundamental en el desarrollo de modelos de redes neuronales convolucionales. Si bien se recomienda como mínimo la división de dos subconjuntos, hacerlo en tres permite una mejor evaluación exhaustiva y objetiva del rendimiento del modelo, contribuyendo así a su optimización y mejora continua.

Como se explica con mayor detalle en el anexo teórico B, en el apartado B.2 Desarrollo de redes neuronales

, el subconjunto de entrenamiento se emplea para ajustar los pesos de la red durante el proceso de aprendizaje. Por su parte, el de validación supervisa el comportamiento del modelo durante el entrenamiento y permite tomar decisiones como el uso de early stopping¹², la elección de hiperparámetros o la detección de posibles problemas de sobreajuste. Por último, el conjunto de prueba proporciona una evaluación final, imparcial e independiente del modelo ya entrenado, midiendo su capacidad real de generalización sobre datos nunca vistos.

La implementación de este procedimiento garantiza la integridad de los resultados obtenidos, al evitar la influencia de los datos empleados durante el proceso de entrenamiento. De este modo,

¹² Técnica de regularización utilizada en entrenamientos de modelos de Machine Learning, monitoreando una métrica seleccionada para la ejecución del entreno cuando ésta deja de mejorar.

se consigue una representación más precisa del rendimiento esperado en escenarios del mundo real.

En nuestro caso, se implementó una estrategia de subdivisión que asignó el 70 % de las imágenes al subconjunto de entrenamiento, el 15 % al de validación y el 15 % restante al de prueba. En el proceso de división, se preservó la proporción de imágenes con y sin presencia animal en los conjuntos de validación y test, manteniendo la misma distribución observada en el conjunto de datos completo.

Tabla 1: Número de imágenes de cada clase en cada división del dataset

Conjunto	Porcentaje de imágenes disponibles	Número total de imágenes	Número de imágenes con presencia animal	Número de imágenes vacías
Entrenamiento	70%	19.995	13.230	6.765
Validación	15%	4.286	2.836	1.450
Prueba	15%	4.286	2.836	1.450

Se evidencia un desbalanceo entre las clases, con una predominancia de casos positivos que alcanzan aproximadamente el 66%, en contraste con los casos negativos que representan el 34% del total.

3.3 Métricas de evaluación

Con el propósito de evaluar la eficacia del modelo de clasificación binaria propuesto, se ha implementado un conjunto de métricas que facilitan la observación de su comportamiento desde múltiples perspectivas.

- **Accuracy:** mide el porcentaje global de predicciones correctas.
- **Loss:** refleja el error durante el entrenamiento y la validación
- **Precision:** ofrece información más detallada sobre la calidad de las predicciones positivas, indica la proporción de predicciones positivas que resultaron ser correctas.
- **Recall:** también ofrece información detallada sobre la calidad de las predicciones positivas, en este caso, la proporción de instancias positivas reales que fueron identificadas correctamente.
- **Specificity:** también ofrece información detallada sobre la calidad de las predicciones negativas, en este caso, la proporción de instancias negativas reales que fueron identificadas correctamente.
- **F1-Score:** integra ambas métricas (precision y recall) en una única medida equilibrada.
- **Matriz de confusión:** para visualizar de forma detallada la distribución de aciertos y errores en las predicciones.
- **AUC:** Área Bajo la Curva ROC evalúa la capacidad del modelo para distinguir entre clases.
- **Curva ROC:** complementada con la métrica anterior, en este caso ilustra gráficamente la capacidad de distinguir entre clases.

Todas estas métricas se encuentran mejor definidas en el Anexo teórico B, en su apartado B.2.1
Métricas de evaluación

Capítulo 4 Metodología

En el presente capítulo, se procederá a la exposición de la metodología empleada en el desarrollo del proyecto, así como las distintas fases que han sido planteadas y las arquitecturas implementadas para la implementación del modelo, con el propósito de alcanzar el objetivo propuesto.

Anexo teórico A Para una mejor comprensión del resto del presente apartado, se recomienda la lectura de todos los anexos teóricos, haciendo especial mención a los anexos teóricos B Redes neuronales y anexo teórico C Redes neuronales convolucionales

Para cumplir el objetivo principal de este proyecto, que consiste en implementar una herramienta con técnicas de deep learning capaz de clasificar imágenes con o sin animales, se ha desarrollado una CNN clasificadora a la que hemos denominado MegaClassifier_C.

Como se ha mencionado anteriormente, MegaDetector es una herramienta muy potente que, a pesar de ser un modelo de detección, podría adaptarse a nuestro caso mediante la interpretación de su salida. Esto es lo que se ha hecho para tener una referencia de calidad.

De forma secundaria, se ha intentado implementar una combinación de ambas herramientas, donde MegaDetector funciona como un preprocesador de las imágenes que serán la entrada de nuestro clasificador. Según cómo se trate esta entrada, se diseñaron dos variantes (MegaClassifier_A y MegaClassifier_B) que se expondrán de forma anecdótica para contrastar aún más los resultados de nuestro principal clasificador.

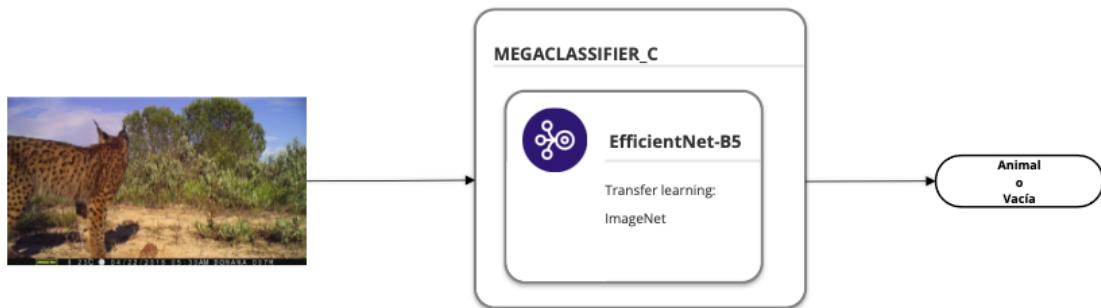
4.1 Arquitectura de la red

A.1.1 Como se acaba de mencionar, en el marco de este proyecto se gestionan tres arquitecturas distintas. La primera y principal de ellas se fundamentada en la arquitectura EfficientNet-B5, la cual se ha detallado en el anexo teórico C en su apartado C.3.4 Arquitectura utilizada en este trabajo

.

Originalmente, esta arquitectura ha sido concebida para clasificar un total de 1.000 clases, lo cual, a priori, no la hace pertinente para la problemática que nos ocupa. Por consiguiente, se ha visto en la necesidad de realizar una serie de modificaciones, que ha implicado la eliminación de lo que se denomina *cabeza de red*.

El procedimiento descrito se centra en la eliminación de la capa densa con función de activación softmax. Esta capa densa ha sido diseñada con el propósito de clasificar todas las clases de



ImageNet. Al eliminar esta capa, se prescinde de la parte encargada de la clasificación, conservando únicamente la parte convolucional, capaz de extraer las características. Para adaptar este enfoque al caso en cuestión, se ha introducido una nueva capa densa con una única salida y una función de activación sigmoide, ampliamente utilizada en aplicaciones de clasificación binaria (ver Figura 9: Esquema del modelo principal MegaClassifier_C).

Figura 9: Esquema del modelo principal *MegaClassifier_C*

4.2 Hiperparámetros

A.1.2 Como se explica en el anexo teórico B en el apartado B.2.3 Inicialización de los pesos

a la hora de realizar un entrenamiento, se requiere la inicialización de los pesos. Entre las diversas técnicas presentadas en el apartado anterior, se ha seleccionado la opción de transfer learning.

Esta técnica se fundamenta en la reutilización del conocimiento adquirido por un modelo previamente entrenado en una tarea amplia y general, como se evidencia en el caso de la clasificación del conjunto de datos ImageNet. Al aprovechar los pesos preentrenados, el modelo puede iniciar con una representación visual robusta, lo que reduce el tiempo de entrenamiento y el riesgo de sobreajuste, y facilita una convergencia más rápida hacia una solución efectiva en la tarea objetivo.

En el contexto de facilitar la implementación de este proyecto como una aproximación inicial para abordar desafíos relacionados con el uso de CNN o como un referente para la estructuración de un proyecto de Deep Learning, se determinó preservar una metodología de versionado que permitiera, en cada iteración, la selección u optimización de alguno de los hiperparámetros.

4.2.1 Batch size

En esta primera versión del entreno de nuestra CNN, se implementa un proceso de prueba que abarca la evaluación de diferentes tamaños de lote. Los valores que se han sometido a prueba comprenden de [16, 32, 64, 128], siendo estos valores comunes y ampliamente utilizados.

La selección óptima del tamaño del conjunto de datos es crucial para el rendimiento de una red neuronal convolucional, ya que determina el equilibrio entre la precisión y la eficiencia computacional. Un valor reducido, como 16, puede ofrecer una mejor generalización y escapar de los mínimos locales, mientras que valores mayores, como 64 o 128, aceleran el entrenamiento y estabilizan el gradiente. El objetivo es identificar el punto de equilibrio entre la velocidad de entrenamiento y la calidad del aprendizaje, de modo que se maximice el rendimiento del modelo.

4.2.2 Algoritmo de optimización

La elección del optimizador más apropiado para una CNN constituye una determinación de suma importancia que incide de manera directa en la convergencia, la estabilidad y el desempeño final del modelo, como se detalla en el anexo teórico B, apartado B.2.4 Algoritmos de optimización

. Los algoritmos probados han sido Adam, RMSprop y SGD, basándose en las diferentes ventajas que ofrecen: Adam se erige como una entidad que fusiona de manera sinérgica los atributos del momento adaptativo y la normalización del gradiente, logrando así un equilibrio dinámico entre la celeridad y la exactitud; RMSprop se distingue por su capacidad para gestionar gradientes cambiantes mediante tasas de aprendizaje adaptativas; por otro lado, a pesar de su aparente simplicidad, SGD puede alcanzar mejores mínimos globales en ciertos escenarios.

4.2.3 Función de pérdida

A.1.3 Fundamentado con los conceptos que aparecen en el anexo teórico B, en su apartado B.2.2 Estimación del error

y con el propósito de optimizar el desempeño del modelo y adaptarlo de manera efectiva al desequilibrio presente en el conjunto de datos, se ha determinado llevar a cabo experimentos con diversas funciones de pérdida durante el proceso de entrenamiento. Además de la función de pérdida binaria estándar BinaryCrossentropy, se ha incorporado el uso de BinaryFocalCrossentropy, reconocida por su capacidad para enfocarse en ejemplos complejos y mitigar el impacto del desequilibrio entre las clases. Asimismo, se ha evaluado el uso de BinaryCrossentropy en conjunto con un ajuste de pesos por clase, parámetro conocido como class_weight, lo cual permite penalizar más los errores en la clase minoritaria. Esta estrategia comparativa tiene como objetivo determinar cuál de estas opciones ofrece el mejor equilibrio entre precisión y sensibilidad, maximizando así el rendimiento del modelo en un entorno de clasificación binaria desequilibrada.

4.2.4 Valores numéricos de los hiperparámetros

Con el fin de maximizar el rendimiento del modelo, resulta fundamental realizar una adecuada selección de los hiperparámetros que controlan el proceso de entrenamiento. En este trabajo, se ha prestado especial atención al valor del learning rate del optimizador, ya que este parámetro constituye un elemento determinante en la velocidad de aprendizaje del modelo y puede incidir de manera directa en la estabilidad y la capacidad de convergencia. Además, se ha seleccionado

la función de pérdida BinaryFocalCrossentropy, que ha demostrado ser la más efectiva en todas las arquitecturas, y se han incluido como hiperparámetros a optimizar sus dos componentes clave: alpha, que controla la ponderación entre clases, y gamma, que regula el enfoque del modelo hacia los ejemplos más difíciles. La optimización de estos parámetros resulta imperativa para la construcción de modelos más robustos, precisos y adaptados a la naturaleza del problema en cuestión.

4.2.5 Data augmentation

Se ha implementado el uso de técnicas de augmentation de datos con el propósito de incrementar la capacidad de generalización del modelo y minimizar el riesgo de sobreajuste. Este procedimiento implica la aplicación de transformaciones aleatorias a las imágenes utilizadas para el entrenamiento, tales como giros, variaciones de brillo, recortes o escalados. De esta manera, se generan nuevas versiones sintéticas de las muestras originales. La implementación de un conjunto de datos más heterogéneo durante el entrenamiento fomenta el aprendizaje de características más robustas e invariantes por parte del modelo. Esta capacidad resulta particularmente beneficiosa en aplicaciones como el fototrampeo, donde las condiciones de iluminación, ángulos y fondos pueden presentar variaciones significativas. En consecuencia, se evidencia un incremento en la capacidad del modelo para adaptarse a nuevas imágenes sin perder precisión.

4.2.6 Early Stopping

En las fases más avanzadas del desarrollo se ha introducido la técnica de early stopping con el propósito de evitar el sobreentrenamiento y seleccionar el punto óptimo del aprendizaje antes de que el modelo comience a degradar su rendimiento en el conjunto de validación. En la implementación de esta técnica se han explorado diversos criterios de parada temprana, incluyendo las métricas loss, AUC, precision y recall. El uso de loss permite identificar el momento en que el modelo deja de minimizar el error global, mientras que AUC ofrece una visión más completa del comportamiento del modelo en términos de separabilidad entre clases, lo cual resulta especialmente útil en problemas con cierto desequilibrio. Además, se han considerado las métricas precision y recall, dada su importancia en tareas de clasificación binaria como la presente, donde es esencial reducir tanto los falsos positivos como los falsos negativos. La comparación del comportamiento del early stopping en función de estas métricas permite la elección del enfoque más alineado con los objetivos del problema.

4.2.7 Fine Tuning

Por último, se ha hecho uso de la técnica conocida como fine tuning, un paso fundamental en el contexto del transfer learning, que consiste en ajustar los pesos de las capas previamente entrenadas para adaptarlas mejor a la tarea específica del proyecto. Para ello, se ha evaluado el rendimiento del modelo al descongelar distintos bloques de capas de la arquitectura EfficientNet-B5: 20, 40, 80 capas y, finalmente, la totalidad del modelo. Esta progresión escalonada permite observar el impacto que tiene la liberación de distintos niveles de

abstracción en la red, desde características más generales (como bordes y texturas) hasta otras más específicas. El propósito es hallar un equilibrio entre la capacidad de aprendizaje y el riesgo de sobreajuste, ya que al desbloquear más capas se incrementa el número de parámetros entrenables y, en consecuencia, la complejidad del modelo. Esta estrategia permite maximizar la especialización del modelo en el dominio concreto del conjunto de datos, partiendo de una base robusta ya preentrenada con ImageNet.

4.3 Fase de entrenamiento

En este apartado se presentan los resultados obtenidos durante el entrenamiento de cada versión, lo que permite analizar la evolución del rendimiento del modelo y el impacto de las diferentes decisiones tomadas.

Es preciso considerar que, en las primeras versiones del modelo, el enfoque se ha centrado en aislar y analizar de forma individual cada uno de los componentes clave del sistema, tales como el tamaño de lote, la función de pérdida o el ajuste de hiperparámetros, entre otros. El propósito de estas etapas iniciales no radicaba en optimizar directamente el rendimiento final, sino más bien en comprender el impacto de cada elemento en el comportamiento del modelo y establecer una base sólida para versiones posteriores. No es hasta la versión 5 cuando se integran las decisiones más prometedoras tomadas en versiones anteriores, combinando técnicas como el data augmentation o el uso de funciones de pérdida robustas, para construir un modelo más completo. A partir de este punto, el enfoque se centra en refinar el rendimiento global mediante mejoras acumulativas, tales como el uso de early stopping o el fine-tuning, lo que permite una evolución más coherente y sostenida en las métricas de evaluación.

Como se ha mencionado, los resultados del uso de la herramienta MegaDetector se utilizarán tanto como referencia para saber cómo de bueno resulta el rendimiento de la CNN que implementemos, como de preprocesamiento de los datos de entrada de la CNNs experimentales que se han propuesto de forma paralela. Es sabido que este modelo funciona con un umbral de confianza para generar las detecciones, investigando en su repositorio se ha podido recopilar los siguientes valores: cuando el modelo es ejecutado sin especificar un valor, por defecto utiliza 0.1, en los ficheros JSON que genera se especifican dos umbrales que vienen denominados como *típico*, con un valor de 0.2; y *conservador*, con un valor de 0.05.

Además de estos valores, para poder utilizar los datos como entrada para las CNNs experimentales, donde se ha buscado un umbral muy generalista capaz de cubrir todas las fotos con animales del conjunto de entrenamiento. Esta determinación se fundamenta en la necesidad imperativa de asegurar que ningún ejemplo positivo relevante sea excluido del proceso de entrenamiento como resultado de un error en la detección previa, esto a su vez conlleva asumir un gran número de falsos positivos que trataran de corregir las CNNs experimentales. Como estos modelos pueden desviarse del objetivo principal, se ha decidido darles menos importancia en este documento y hacer referencia a ellos solo cuando sea relevante.

4.3.1 Métricas de referencia de MegaDetector

Ejecutando MegaDetector con los diferentes umbrales anteriormente mencionados, sobre el conjunto de entrenamiento, se han podido obtener los siguientes resultados mostrados en la Tabla 2:

Tabla 2: Cobertura de positivos de MegaDetector con diferentes umbrales

Umbral	Positivos en el conjunto de entrenamiento	Positivos detectados	Porcentaje de cobertura
0.2		13.052	98.65%
0.1		13.098	99.00%
0.05	13.230	13.136	99.29%
0.0015		13.230	100%

Con base en los datos generados por la implementación de MegaDetector sobre el conjunto completo de datos con el umbral establecido, es factible generar métricas de referencia. En el caso de que el detector genere detecciones de una imagen con presencia animal, se puede contabilizar como un verdadero positivo; cuando no genera detecciones, se puede contabilizar como un verdadero negativo; siguiendo con este criterio, cuando genera detecciones de una imagen vacía, se encuentra en el caso de un falso positivo y, por último, cuando no genera detecciones de una imagen con presencia animal, se puede contabilizar como un falso negativo.

A.1.4 Bajo el criterio expuesto y con las métricas presentadas en el anexo teórico B apartado B.2.1 Métricas de evaluación

se ha creado el sistema de referencia durante la fase de entrenamiento, compuesto por:

Las matrices de confusión ideales del conjunto de entrenamiento y validación, representadas en la Figura 10 y Figura 11, respectivamente.

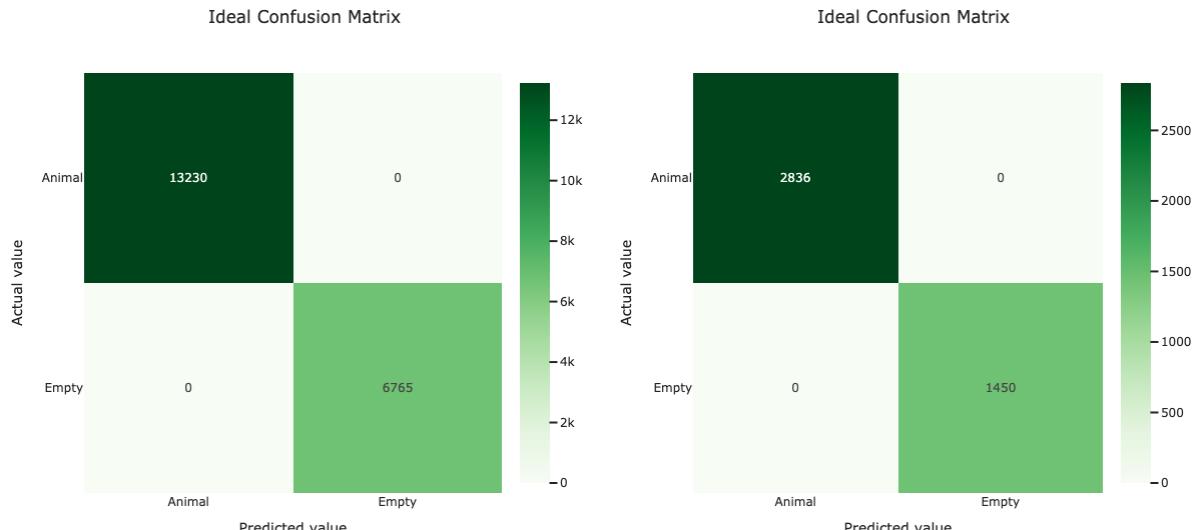


Figura 10: Matriz de confusión ideal de train

Figura 11: Matriz de confusión ideal de validation

Las matrices de confusión en ejecuciones con umbral por defecto sobre el conjunto de entrenamiento representada en la Figura 13 y sobre el conjunto de validación representada en la

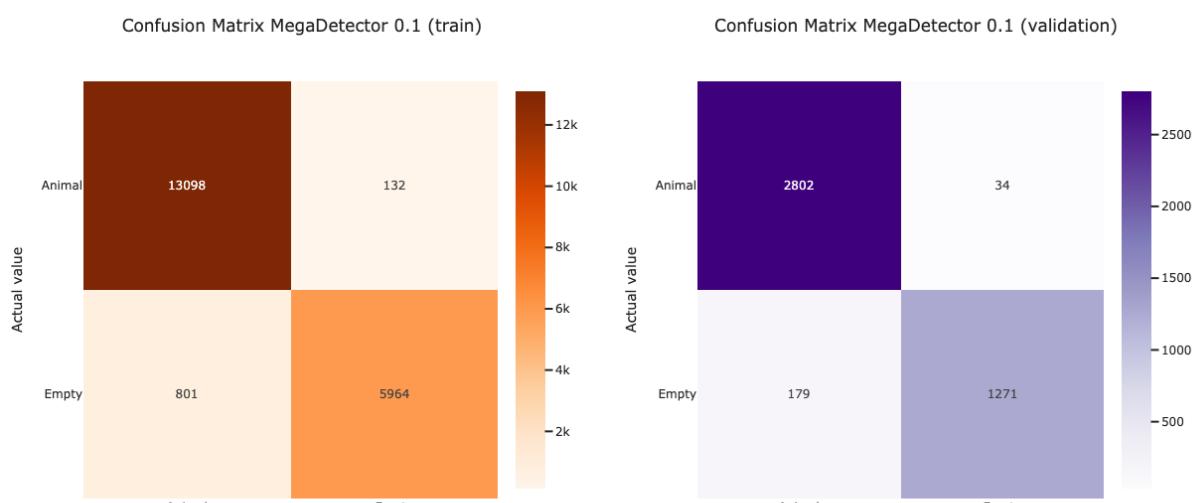


Figura 13: Matriz de confusión de train de
MegaDetector umbral=0.1

Figura 12: Matriz de confusión de validation de
MegaDetector umbral=0.1

Figura 13.

Tabla 3: Métricas de MegaDetector sobre validation

Modelo	Accuracy	Loss	Precision	Recall	Specificity	F1-Score	AUC
MD_0.2	0,9620	-	0,9610	0,9824	0,9221	0,9716	-
MD_0.1	0,9503	-	0,9400	0,9880	0,8766	0,9634	-
MD_0.05	0,9326	-	0,9136	0,9919	0,8166	0,9511	-

A.1.5 Entrenamiento de MegaClassifier

A continuación, describimos el proceso de entrenamiento del modelo MegaClassifier_C. El objetivo es analizar la capacidad del modelo de generalizar, verificar la estabilidad de su entrenamiento y comparar las diferentes configuraciones con el fin de encontrar el mejor desempeño posible.

En la versión 1, se evaluaron diferentes tamaños de lote [16, 32, 64 y 128] con el objetivo de determinar aquel que ofrecía un mejor equilibrio entre estabilidad en el entrenamiento y rendimiento en validación.

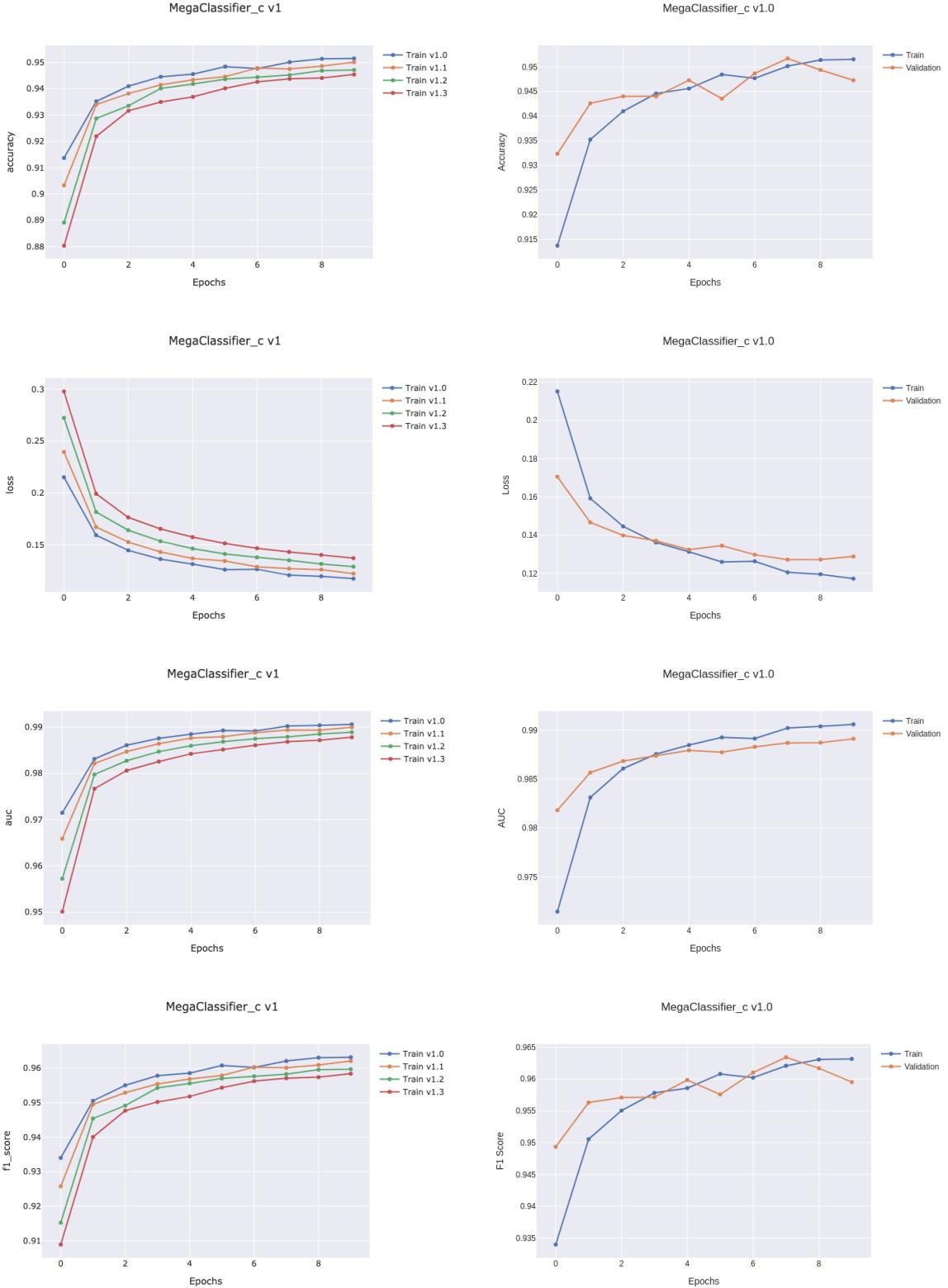


Figura 14: Gráficas de entrenamiento de la versión 1

Los resultados que podemos ver en la Figura 14 evidencian que escoger un batch size de 16 ofrece un equilibrio adecuado entre velocidad de convergencia, estabilidad del entrenamiento y capacidad de generalización. Aunque un batch size menor implica un número mayor de pasos de optimización y puede conllevar un incremento en el tiempo de entrenamiento, la mejora en el

ajuste del modelo (reflejada en la ausencia de sobreajuste pronunciado y en métricas de validación competitivas) justifica la elección de 16. Así, esta primera versión sienta las bases para las siguientes etapas de ajuste, en las que se explorarán optimizadores, funciones de pérdida, hiperparámetros y otras técnicas para optimizar aún más el rendimiento.

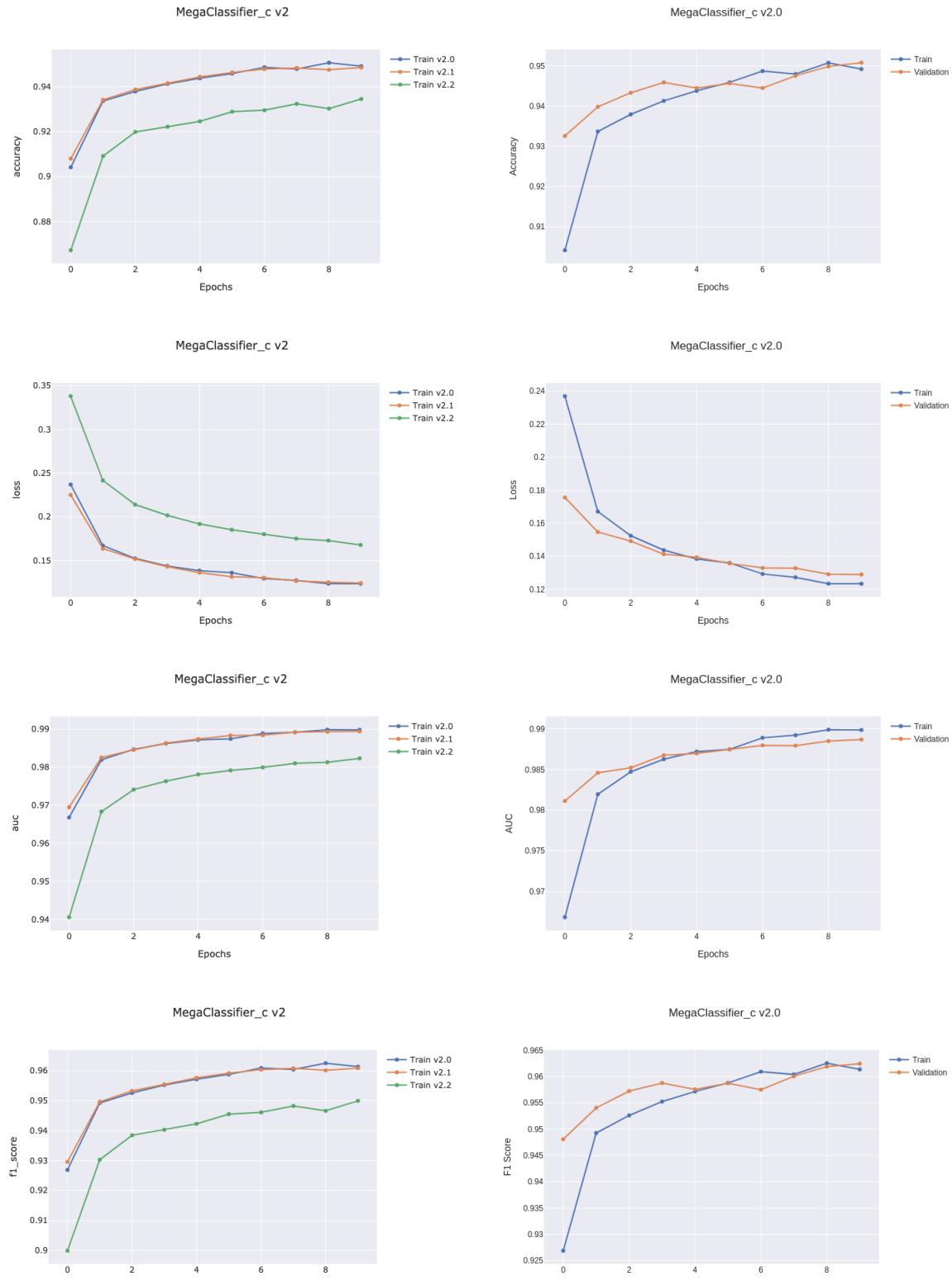


Figura 15: Gráficas de entrenamiento de la versión 2

En la versión 2, el objetivo principal fue determinar el optimizador más apropiado para esta arquitectura, probando Adam, RMSprop y SGD bajo las mismas condiciones de entrenamiento (incluyendo el batch size de 16 ya seleccionado). Al comparar las métricas de accuracy, F1-score, AUC y la evolución de la loss, se observó que Adam presentaba una convergencia más estable y con un mejor equilibrio entre entrenamiento y validación, mientras que RMSprop tendió a mostrar fluctuaciones mayores y SGD necesitó más tiempo para alcanzar resultados similares. Aunque SGD con momento y un ajuste fino del learning rate podría mejorar sus prestaciones, el desempeño inmediato de Adam resultó más consistente, lo que justificó la decisión de adoptarlo como optimizador base para las versiones posteriores.

Aunque, como se puede apreciar en las gráficas de la Figura 15, los resultados con RMSprop y Adam son en general muy parecidos, se aprecian ligeras ventajas a favor de Adam en términos de estabilidad y rapidez de convergencia. Con Adam, el valor de loss tiende a descender de forma más consistente en etapas tempranas, y las curvas de accuracy y F1-score se estabilizan antes que con RMSprop. Además, Adam combina la idea de acumulación de gradientes con un ajuste adaptativo de la tasa de aprendizaje para cada parámetro, lo que puede explicar por qué reacciona mejor a variaciones en los datos y evita en mayor medida las oscilaciones que todavía se observan, aunque mínimas, con RMSprop. Estas diferencias, aunque sutiles, inclinan la balanza hacia la adopción de Adam para optimizar el proceso de entrenamiento.

En esta versión 3, se exploraron distintas funciones de pérdida para abordar la clasificación binaria en condiciones de posible desbalance de clases: Binary Crossentropy estándar, Binary Crossentropy con pesos y Binary Focal Crossentropy. En las gráficas obtenidas (ver Figura 16), se aprecia que la versión con Focal Loss alcanza un mejor desempeño global, reflejándose en métricas como el F1-score y el AUC, además de una convergencia más estable. Al poner mayor énfasis en los ejemplos difíciles de clasificar, la Focal Loss consigue reducir la influencia de las muestras mayoritarias y mejora la sensibilidad hacia la clase minoritaria. Mientras que las otras dos funciones de pérdida también ofrecen resultados aceptables, sus curvas muestran ligeras desventajas frente a Focal, sobre todo en escenarios donde el desbalance puede afectar la calidad de la clasificación. Por ello, se eligió Binary Focal Crossentropy como la mejor opción para las versiones subsiguientes.

En la versión 4 del modelo, se inicia el proceso de ajuste fino de hiperparámetros con el objetivo de maximizar el rendimiento de la configuración seleccionada como base en versiones anteriores. Teniendo en cuenta que tanto el optimizador Adam como la función de pérdida BinaryFocalCrossentropy han demostrado ser las opciones más efectivas, esta fase se centra en optimizar sus parámetros internos. En particular, se busca encontrar el valor óptimo del learning rate para Adam, así como los valores de alpha y gamma para BinaryFocalCrossentropy, que controlan, respectivamente, el peso asignado a la clase minoritaria y el grado de penalización a las predicciones incorrectas. Este procedimiento resulta fundamental para potenciar la capacidad de generalización del modelo y su sensibilidad en contextos con datos desequilibrados.

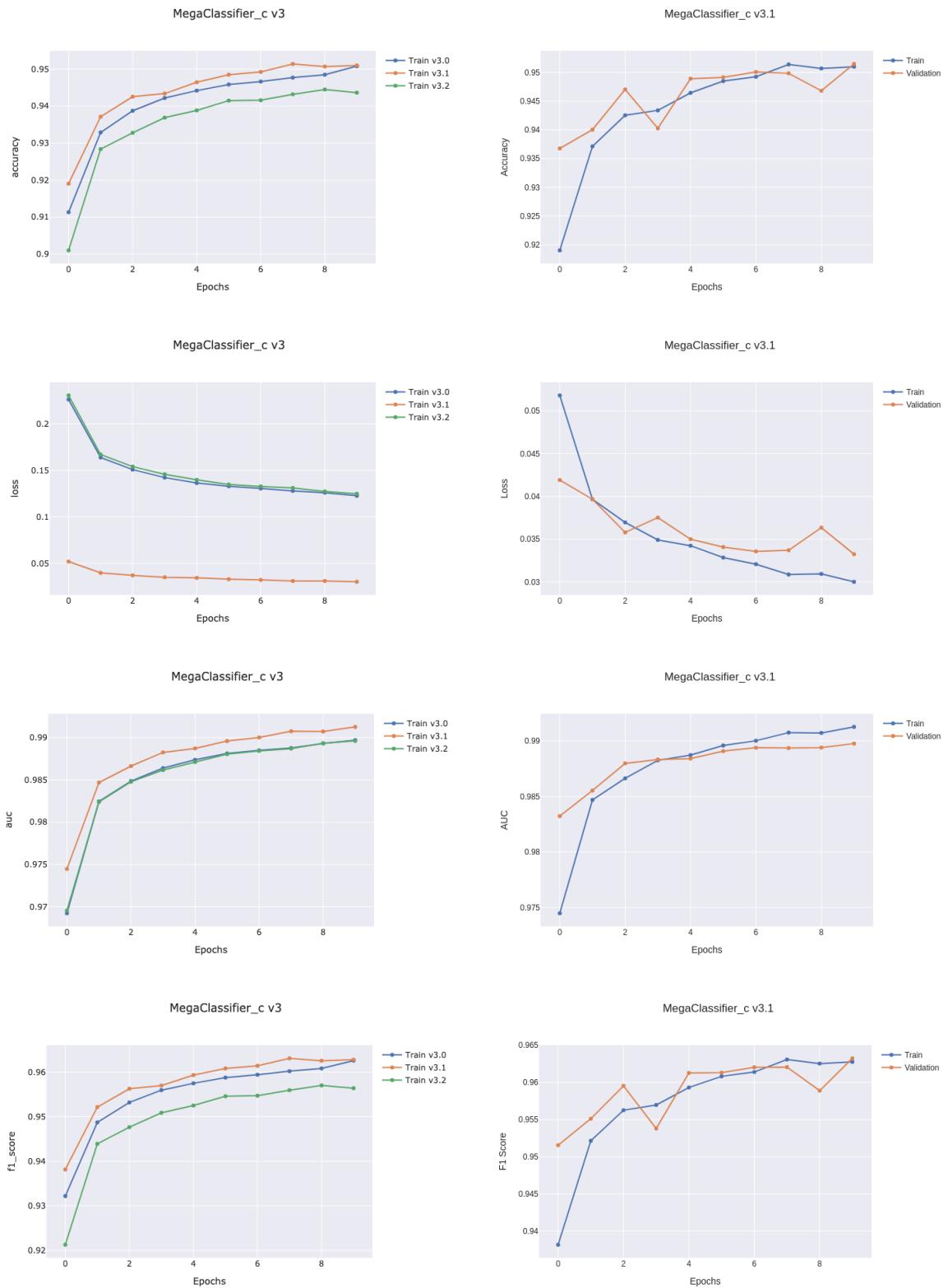


Figura 16: Gráficas de entrenamiento de la versión 3

En el marco del ajuste de hiperparámetros para el entrenamiento de redes neuronales, Leslie N. Smith propuso en su artículo la técnica conocida como Learning Rate Finder (LRFinder) [1]. Este método tiene como objetivo determinar un intervalo apropiado para el learning rate sin recurrir

únicamente a búsquedas exhaustivas (grid search), lo que contribuye significativamente a acelerar la convergencia y mejorar la eficiencia del entrenamiento.

La esencia del LRFinder consiste en realizar un breve entrenamiento del modelo a lo largo de un número limitado de iteraciones, incrementando de forma progresiva el valor del learning rate en cada paso, partiendo desde un valor inicial mínimo. Durante este proceso, se registra la evolución de la función de pérdida con respecto al learning rate. Una vez finalizado, se traza una gráfica que relaciona la pérdida con el rango de valores de learning rate, como se muestra en la Figura 17. En esta gráfica, se identifica:

- Una región donde la pérdida se mantiene dentro de un rango constante.
- Un punto (o región) a partir del cual la pérdida desciende con rapidez.
- Un punto donde alcanza el valor mínimo.
- Un punto (o región) a partir del cual la pérdida empieza a divergir.

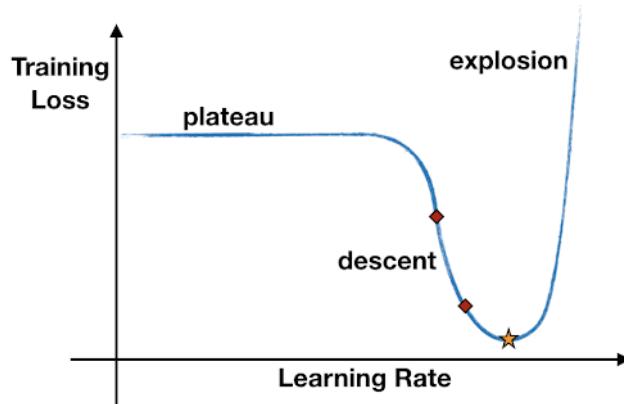


Figura 17: Gráfica teórica de LRFinder¹³

De esta manera, el analista puede seleccionar un intervalo de learning rates en el que la pérdida disminuye de manera estable, escogiendo finalmente el valor más adecuado dentro de dicho intervalo para proceder con el entrenamiento principal. Esta estrategia, introducida por Smith, permite evitar valores demasiado bajos y valores excesivamente altos, logrando así una convergencia más rápida y robusta.

Por consiguiente, al aplicar esta metodología al caso específico en cuestión, se ha obtenido la gráfica que se muestra en la Figura 18, en la que se ha determinado el valor del learning rate como 1×10^{-4} .

¹³ https://blog.dataiku.com/hubfs/training_loss.png

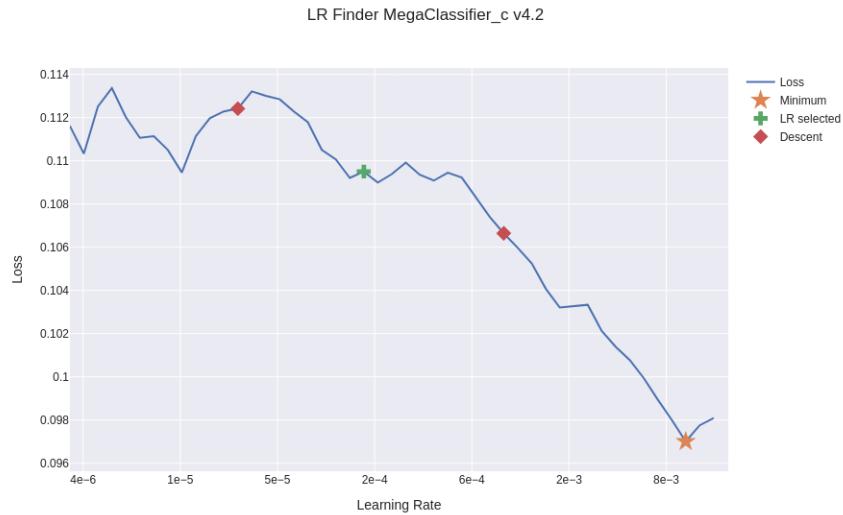


Figura 18: Gráfica del método *LR Finder* aplicada en *MegaClassifier_C*

En lo que respecta al parámetro α de la función de pérdida BinaryFocalCrossentropy, se determinó asignarle un valor constante fundamentado en una fórmula ampliamente empleada en contextos de clasificación desbalanceada. Esta fórmula, que tiene en cuenta la proporción entre la clase mayoritaria y la minoritaria, coincide con los lineamientos descritos en el estudio de 2018 de Buda, Maki y Mazurowski [2], en el cual los autores investigan diferentes estrategias para contrarrestar desequilibrios severos y recomiendan ajustar la ponderación de la función de pérdida de acuerdo con la frecuencia relativa de cada clase. De este modo, se refuerza el impacto de la clase minoritaria y se atenúa la tendencia del modelo a sesgarse hacia la clase más numerosa, aplicando la fórmula:

$$\alpha = \frac{1}{1 + \frac{N_{\text{mayoritaria}}}{N_{\text{minoritaria}}}} \quad (1)$$

Aplicando la fórmula (1) a nuestro caso, obtenemos el valor de $\alpha = 0.3383$.

En cuanto al parámetro γ , que controla la intensidad con la que la función Focal Loss penaliza las predicciones incorrectas con alta confianza, se evaluaron tres valores distintos: 1.0, 2.0 y 3.0. Este hiperparámetro permite enfocar el entrenamiento del modelo en los ejemplos más difíciles, reduciendo la influencia de las predicciones que el modelo ya clasifica correctamente con alta probabilidad.

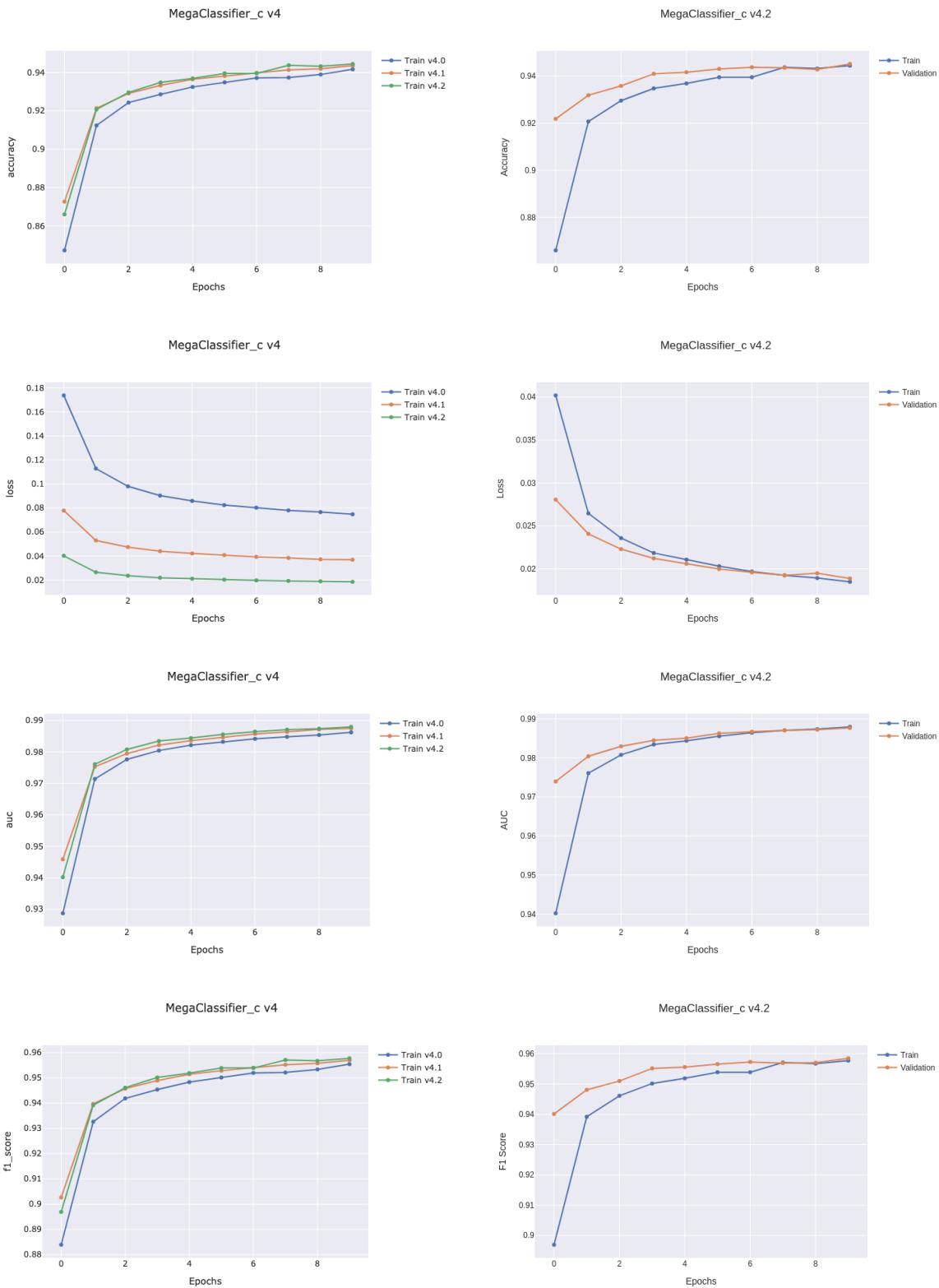


Figura 19: Gráficas de entrenamiento de la versión 4

La versión 4 introduce por primera vez el conjunto de hiperparámetros considerados óptimos. Estos ajustes muestran ya cierto potencial de mejora en las métricas de clasificación, aunque con margen de refinamiento adicional. Como parte de este proceso, se experimentó con distintos

valores de γ , observándose que la versión 4.2, al adoptar $\gamma = 3.0$, resulta la más sólida: penaliza de forma más intensa los falsos negativos, lo que ayuda a mantener un mayor equilibrio entre precisión y sensibilidad, reforzando así la robustez del modelo frente al desbalance. Los resultados visibles en la Figura 19, indican que, en esta instancia, su efecto es beneficioso y se traduce en mejores valores de F1 y AUC en comparación con las pruebas anteriores.

En este punto deberíamos comenzar a ver el potencial del modelo, ya que se han seleccionado los hiperparámetros óptimos en función de los datos de entrenamiento y validación. Para contextualizar los avances de rendimiento de nuestro modelo, se han generado los siguientes datos sobre el conjunto de validación: matrices de confusión del modelo de referencia con su umbral por defecto y MegaClassifier_C v4.2 (ver Figura 20), además de las métricas anteriormente expuestas, como son accuracy, loss, presicion, recall, etc (ver Tabla 4).

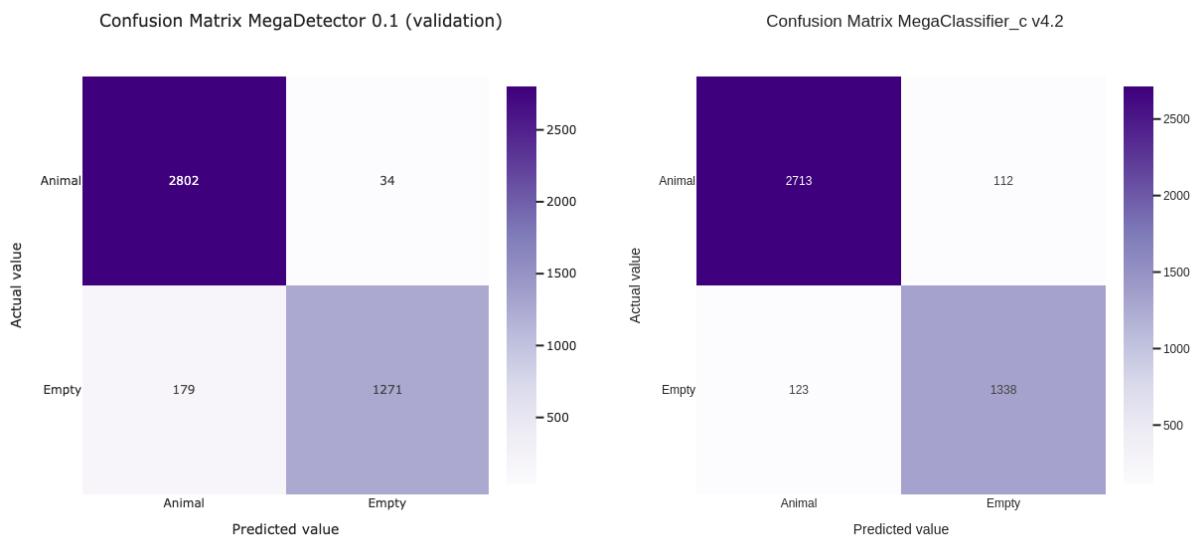


Figura 20: Comparativa de matriz de confusión de MegaDetector con umbral por defecto y MegaClassifier_C en su versión v4.2

Tabla 4: Comparativa MegaDetector y MegaClassifier_C sobre el conjunto de validación

Modelo	Accuracy	Loss	Precision	Recall	Specificity	F1-Score	AUC
MD_0.1	0,9503	-	0,9400	0,9880	0,8766	0,9634	-
MC_v4.2	0,9452	0,0945	0,9566	0,9604	0,9158	0,9585	0,9877

Al comparar los resultados en el conjunto de validación, se observa que nuestro modelo propuesto ha logrado igualar e incluso superar varios de los indicadores de calidad de MD, especialmente en lo que se refiere a precisión (0.9566 vs. 0.9400), specificity (0.9158 vs. 0.8766) y la disponibilidad de un valor de AUC (0.9877). Estos avances sugieren que el sistema está aprendiendo a discriminar mejor la clase vacía, reduciendo falsos positivos.

No obstante, todavía hay aspectos a mejorar, pues MD_0.1 conserva una ligera ventaja en accuracy (0.9503 vs. 0.9452) y recall (0.9880 vs. 0.9604), lo que indica que MC_v4.2 pierde algunos ejemplos de la clase animal y, en consecuencia, su F1-Score es ligeramente inferior (0.9585 vs. 0.9634). Dadas estas cifras, podemos concluir que, aunque estamos a mitad del proceso de optimización, ya se vislumbra el potencial del modelo para acercarse y posiblemente sobrepasar el rendimiento del MegaDetector configurado con su umbral por defecto. Con los ajustes y mejoras previstos en próximas versiones, se espera un equilibrio aún mayor entre precisión y sensibilidad, así como una consolidación del desempeño global del clasificador.

En la versión 5 del modelo se implementaron técnicas de augmentation de datos con el propósito de incrementar la variabilidad del conjunto de entrenamiento y minimizar el riesgo de sobreajuste en etapas posteriores de optimización fina. Esta estrategia busca simular distintas condiciones del dato real, favoreciendo así una mayor capacidad de generalización del modelo sin necesidad de añadir nuevos datos manualmente. De esta manera, se establecen las bases para realizar un ajuste fino más efectivo y seguro, manteniendo la robustez alcanzada en versiones anteriores.

En la versión 5.0 se ha optado por aplicar volteo horizontal y ajustes de brillo en un rango de [0.8, 1.2], estrategia que mejora la capacidad de generalización del modelo de manera más significativa que las demás configuraciones evaluadas. El análisis de las curvas de entrenamiento y validación (ver Figura 21) revela una convergencia más estable, con un descenso más suave en la función de pérdida y valores de accuracy y F1-Score ligeramente superiores respecto a versiones anteriores. Además, el incremento en el AUC indica una mejor separación entre clases, atributo clave en problemas de clasificación binaria desbalanceados. Aunque el beneficio no es drástico, sí se aprecia que el uso de data augmentation comienza a consolidar el desempeño del modelo.

En la versión 6 del modelo se introdujo la técnica de Early Stopping como mecanismo de regularización adicional para evitar el sobreentrenamiento y mejorar la eficiencia del proceso de entrenamiento. El propósito de esta implementación fue inducir la finalización automática del entrenamiento cuando el modelo evidenciara una disminución en la mejora de ciertas métricas críticas, evitando de este modo una pérdida en la generalización. Se evaluaron diversos criterios de monitoreo, tales como la pérdida de validación, con el propósito de minimizarla, así como métricas orientadas a la calidad del modelo, como AUC, precision y recall, en estos casos buscando su maximización en validación. Esta estrategia permite ajustar dinámicamente el número óptimo de épocas, evitando entrenamientos innecesarios y asegurando que el modelo conserve el mejor rendimiento alcanzado durante el proceso.

Como se observa en la, la versión 6.1 se aprecia que todas las métricas evolucionan de forma coherente, evidenciando un comportamiento paralelo entre los conjuntos de entrenamiento y validación. Las curvas de accuracy, F1-score y AUC muestran trayectorias ascendentes que, aunque ligeramente separadas, mantienen un paralelismo que resulta ideal para reflejar un aprendizaje consistente y equilibrado. Del mismo modo, la función de pérdida presenta un descenso paralelamente similar en ambos conjuntos, con una separación moderada que indica

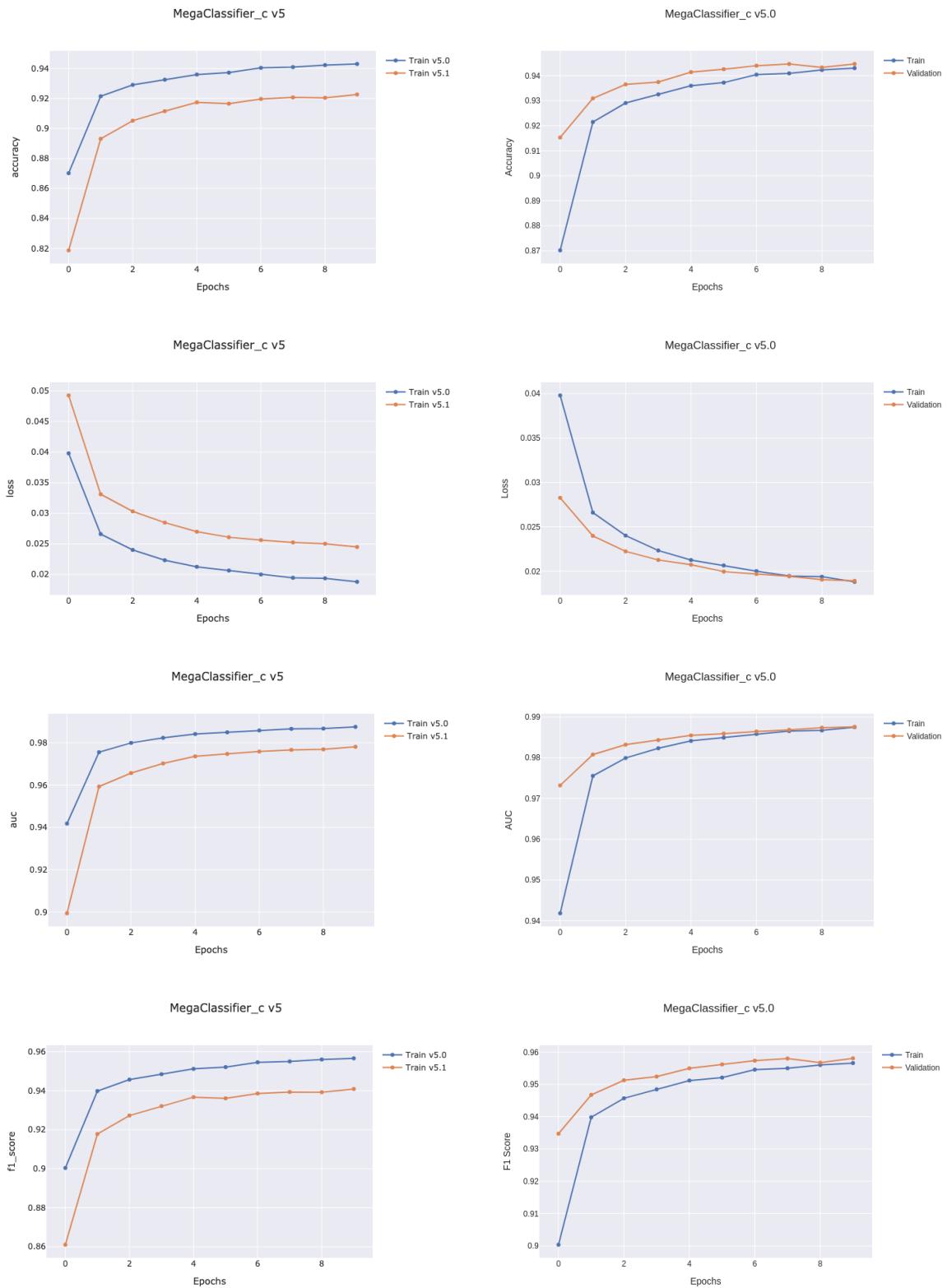


Figura 21: Gráficas de entrenamiento de la versión 5

estabilidad y robustez en la generalización del modelo. Este comportamiento sugiere que el entrenamiento ha alcanzado un punto óptimo, en el cual la evolución de las métricas es confiable y sienta una sólida base para mejoras en futuras iteraciones.

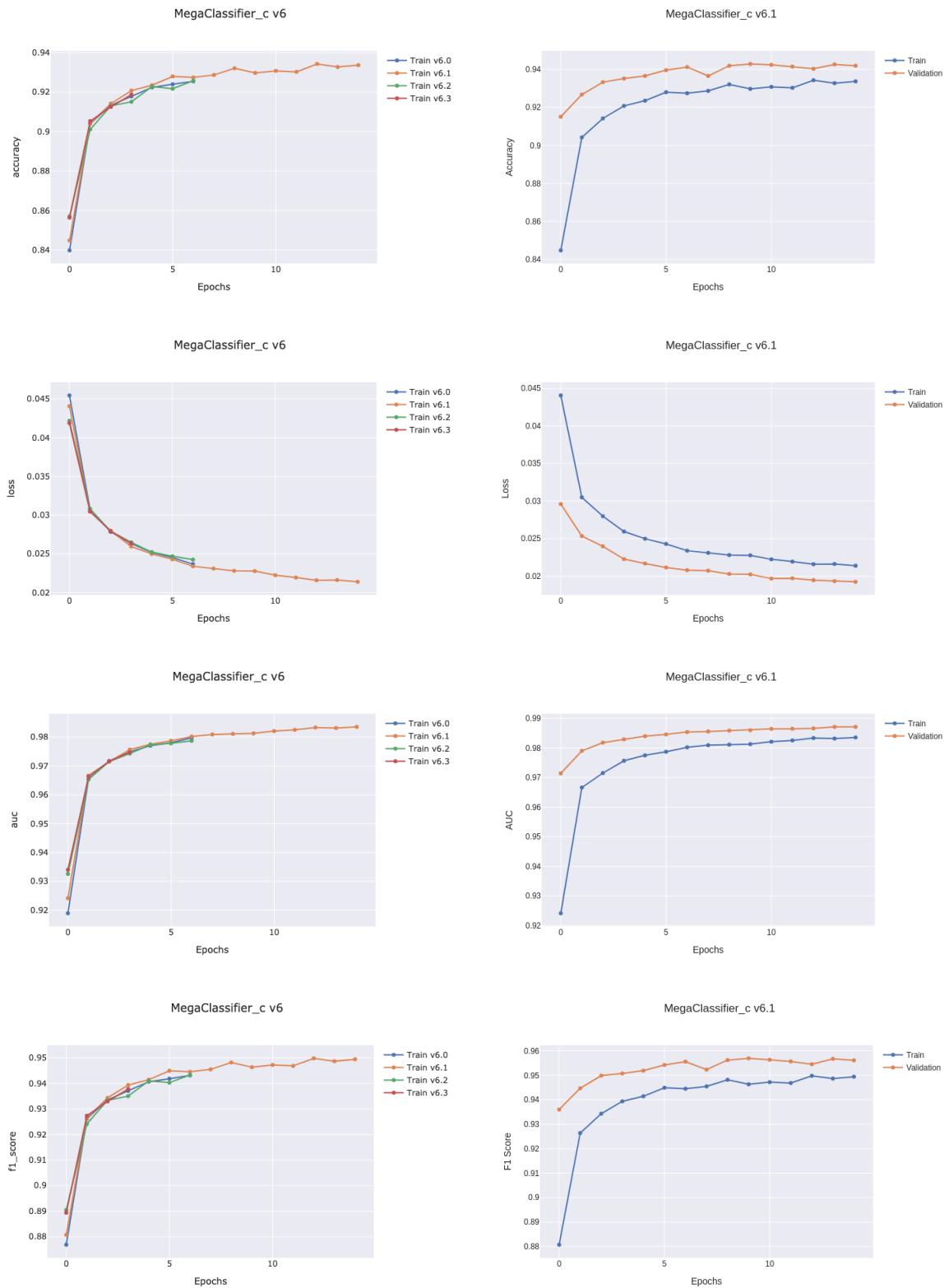


Figura 22: Gráficas de entrenamiento de la versión 6

En la versión 7 del modelo se implementó el proceso de fine-tuning, con el propósito de optimizar el aprendizaje del modelo base previamente entrenado mediante transfer learning. Para ello, se desbloquearon progresivamente distintos bloques de capas del modelo preentrenado,

evaluando el impacto en el rendimiento al permitir que se ajusten sus pesos. Se implementaron cuatro configuraciones: 20 capas, 40 capas, 80 capas y el modelo completo. De manera simultánea, se implementó una disminución en la tasa de aprendizaje en comparación con la fase de entrenamiento inicial, con el propósito de impedir una alteración abrupta de los pesos que ya habían sido optimizados. Esta estrategia permitió analizar el equilibrio entre flexibilidad de ajuste y riesgo de sobreajuste, clave en etapas avanzadas de entrenamiento fino.

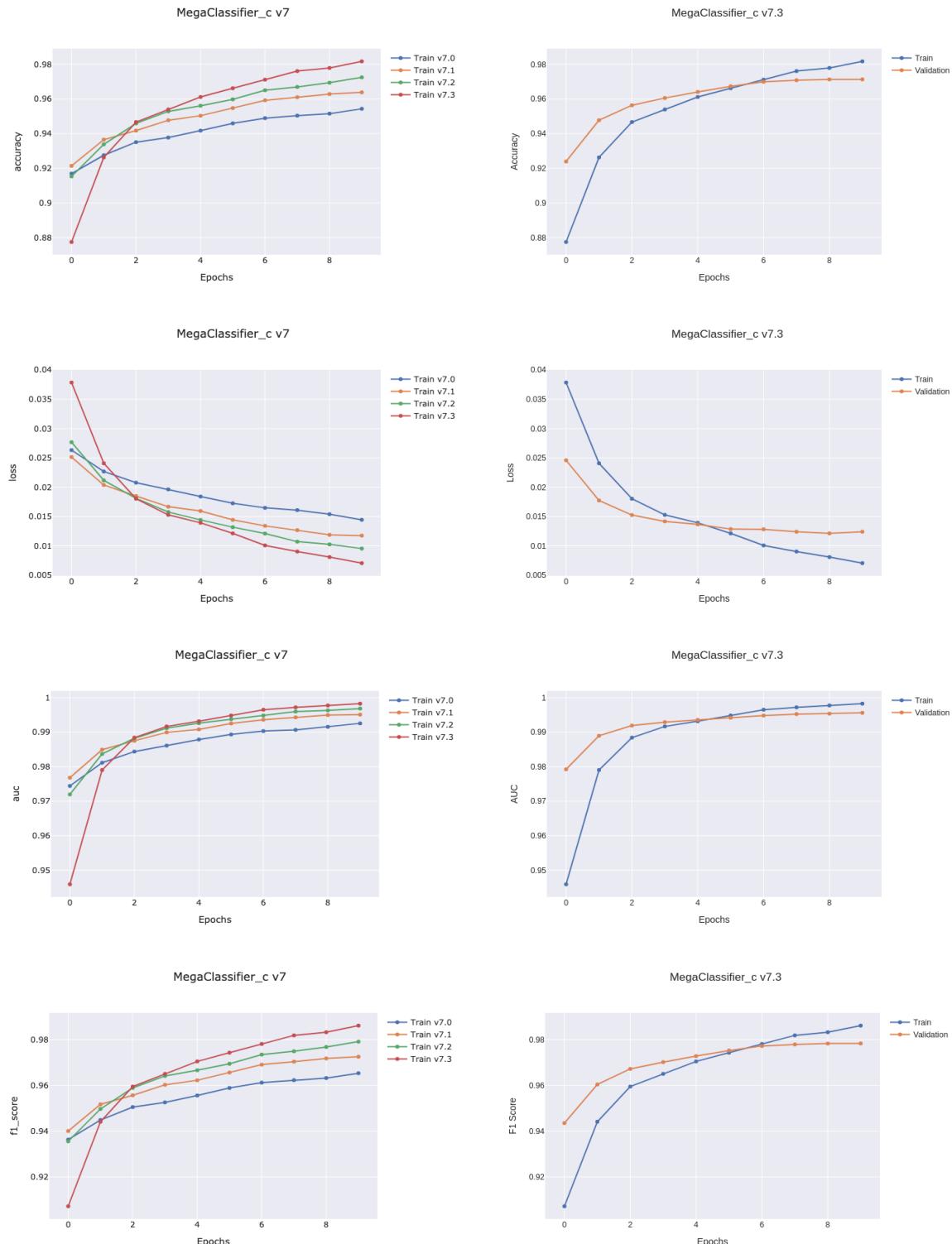


Figura 23: Gráficas de entrenamiento de la versión 7

La versión 7.3 evidencia el máximo rendimiento cuantitativo del modelo hasta el momento, tras la implementación de un ajuste fino exhaustivo. No obstante, el análisis visual de las gráficas sugiere que el modelo comienza a sobre ajustarse, la pérdida se estabiliza o incluso sube levemente, mientras las curvas de rendimiento en entrenamiento siguen creciendo. En consecuencia, se ha optado por avanzar con la versión 8, la cual incorpora la estrategia Early Stopping con el propósito de prevenir que un fine-tuning excesivo comprometa la capacidad de generalización del modelo además de volver a reducir en 10^{-1} el learning rate.

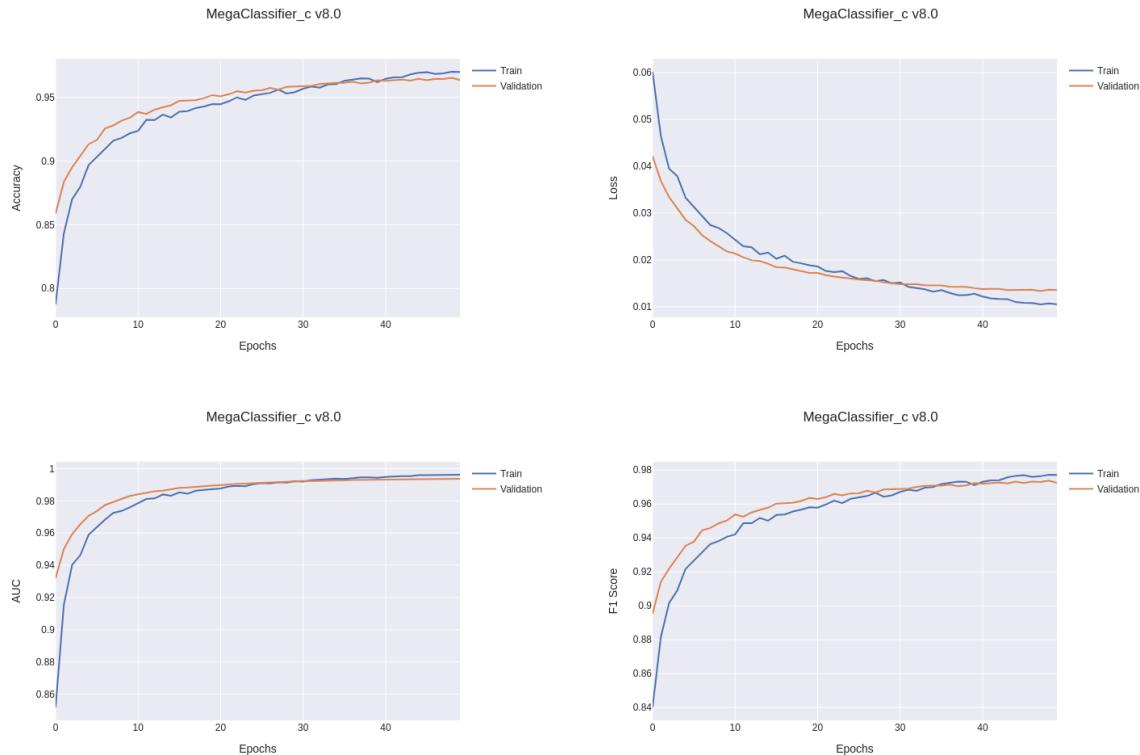


Figura 24: Gráficas de entrenamiento de la versión 8

En la versión final (v8.0) se observa una evolución muy positiva en todas las métricas. La gráfica de accuracy revela un ascenso sostenido en la proporción de predicciones correctas, mientras que el F1-score indica un balance adecuado entre precisión y recall, reflejando que el modelo ha optimizado el compromiso entre ambos. La curva de AUC se mantiene en niveles elevados, confirmando una excelente capacidad para discriminar entre las clases. Por otro lado, la función de pérdida (loss) muestra un descenso regular que se estabiliza en las últimas épocas, evidenciando que el modelo ha alcanzado un entrenamiento sólido y consistente. En conjunto, la paralelidad y la separación moderada entre las curvas de entrenamiento y validación en todas las métricas demuestran que el modelo no solo ha aprendido de manera eficaz, sino que también posee un buen potencial de generalización, consolidando la efectividad del enfoque adoptado en esta versión final.

En este punto, consideramos por entrenado el modelo, denominando a esta versión final **MegaClassifier_C (MC_C)**, por tanto, el siguiente paso es generar métricas sobre el conjunto de

validación para poder compararlas con las obtenidas con MegaDetector, ya que, en caso de ser necesario, podrían introducirse más cambios.

Primeramente, se ha seleccionado el valor de umbral=0.5, ya que es el más popularizado en configuraciones por defecto.

Tabla 5: Comparativa MegaDetector con MegaClassifier_C con umbrales por defecto

Modelo	Umbral	Accuracy	Loss	Precision	Recall	Specificity	F1-Score	AUC
MD	0.1	0,9503	-	0,9400	0,9880	0,8766	0,9634	-
MC_C	0.5	0,9636	0,0136	0,9743	0,9708	0,9493	0,9725	0,9937

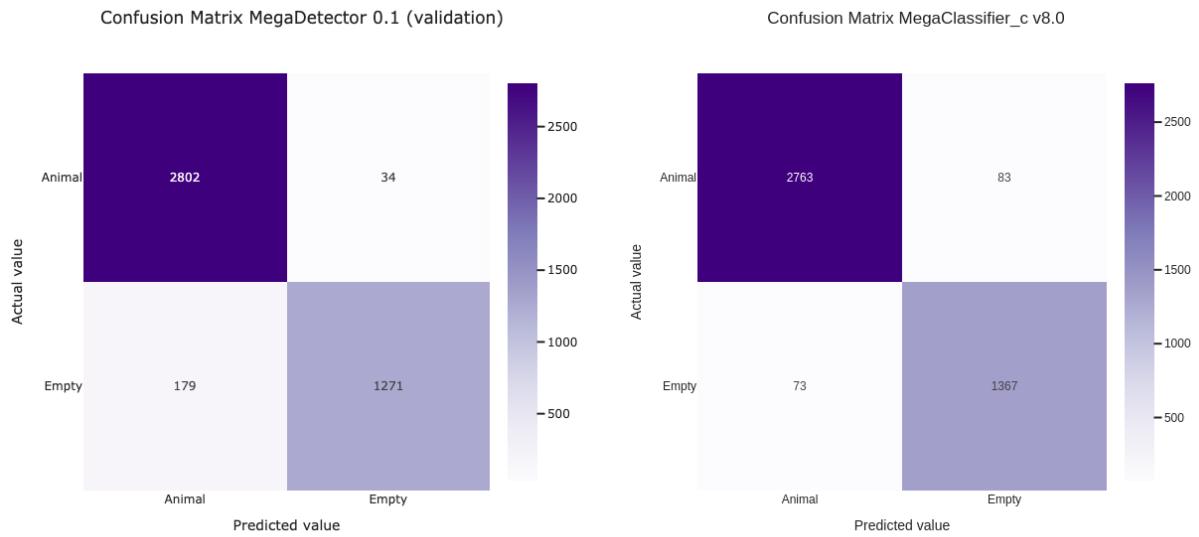


Figura 25: Comparativa de matrices de confusión de validación de MegaDetector y MegaClassifier_C con umbrales por defecto

A partir de la tabla de métricas (ver Tabla 5) y las matrices de confusión (ver Figura 25), se aprecia que MC_C (umbral = 0.5) supera a MegaDetector (MD, umbral = 0.1) en la mayoría de los indicadores. Concretamente, MC_C exhibe mayor accuracy (0.9636 vs. 0.9503), precision (0.9743 vs. 0.9400), specificity (0.9493 vs. 0.8766) y F1-Score (0.9725 vs. 0.9634), además de ofrecer un valor de AUC elevado (0.9937). Estos resultados se corresponden con la matriz de confusión, donde MC_C reduce drásticamente los falsos positivos —73 frente a 179 en MD—, evidenciando una menor confusión al identificar imágenes vacías como animales.

Por otro lado, MD conserva un recall más alto (0.9880 vs. 0.9708), como se aprecia en sus menores falsos negativos (34 en lugar de 83). Esto significa que MegaDetector detecta con más frecuencia los animales presentes, si bien lo hace a costa de mayor número de falsos positivos. En conjunto, MC_C mantiene un equilibrio más robusto entre sensibilidad y precisión, reflejado

en la subida general de métricas y la marcada reducción de falsos positivos, por lo que se perfila como una alternativa sólida que supera al modelo de referencia en la mayoría de los aspectos.

MegaDetector opera con un umbral denominado “Típico” que ha sido diseñado para ofrecer un rendimiento equilibrado en diversas situaciones, optimizando tanto la sensibilidad como la especificidad. En este contexto, resulta fundamental establecer una configuración comparable en MegaClassifier. Para ello, se propone utilizar la curva ROC obtenida a partir del conjunto de validación con el objetivo de determinar el umbral de distancia que minimiza la distancia a la esquina ideal (0,1). Este valor óptimo representa el compromiso perfecto entre la tasa de verdaderos positivos y la de falsos positivos, permitiendo al MegaClassifier adoptar un umbral de operación equivalente al “Típico” de MegaDetector. La implementación de esta estrategia asegurará que el clasificador maximice su capacidad discriminativa, facilitando una comparación directa y un ajuste fino para lograr resultados óptimos en aplicaciones reales.

Tabla 6: Comparativa MegaDetector con MegaClassifier_C con umbrales típicos

Modelo	Umbral	Accuracy	Loss	Precision	Recall	Specificity	F1-Score	AUC
MD	0.2	0,9620	-	0,9610	0,9824	0,9221	0,9716	-
MC_C	0.5334	0,9666	0,0136	0,9690	0,9804	0,9407	0,9746	0,9937

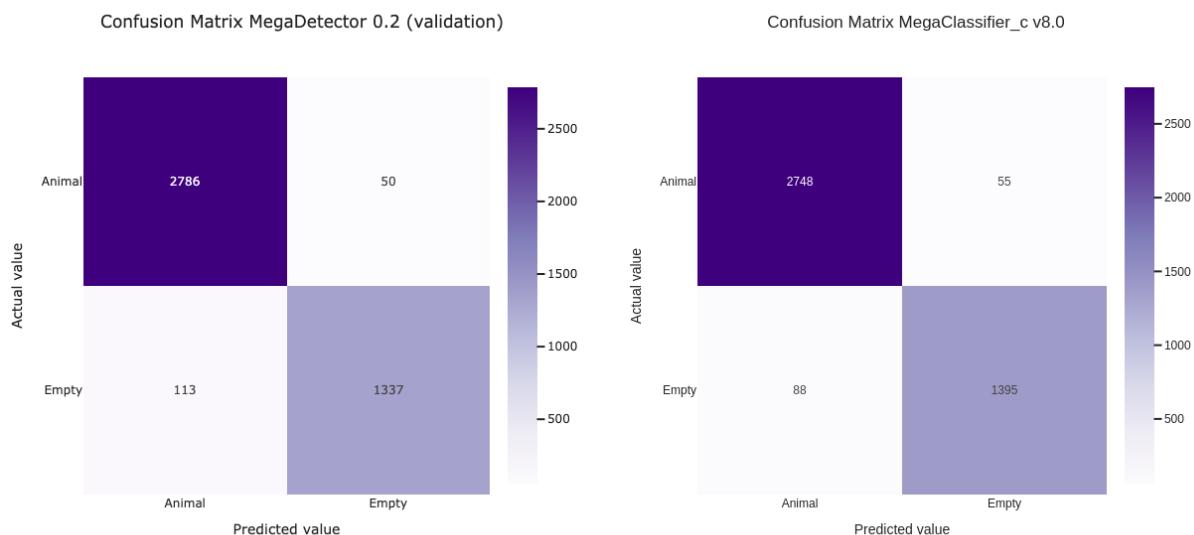


Figura 26: Comparativa de matrices de confusión de validación de MegaDetector y MegaClassifier_C con umbrales típicos

Al comparar las matrices de confusión (ver Figura 26) y las métricas de ambos modelos (ver Tabla 6), se aprecia que MegaClassifier (MC_C), configurado con el umbral derivado de la distancia (0,1) en la curva ROC (0.5334), supera a MegaDetector (MD) (umbral = 0.2) en la mayoría de los indicadores: La accuracy MC_C llega a 0.9666 frente a 0.9620 de MD, lo que implica una proporción ligeramente superior de clasificaciones correctas totales. La precision de MC_C

(0.9690) reduce la cantidad de falsos positivos en relación con los verdaderos positivos, en comparación con MD (0.9610). En cuanto a specificity, la tasa de verdaderos negativos es mayor en MC_C (0.9407) que en MD (0.9221), reflejando mejor discriminación de las imágenes vacías. F1-Score, la media armónica entre precision y recall también favorece a MC_C (0.9746 vs. 0.9716), consolidando un balance más robusto entre ambas. El valor de AUC=0.9937 de MegaClassifier, confirma una elevada capacidad discriminativa.

Desde las matrices de confusión se constata que, si bien MD consigue un recall marginalmente más alto (0.9824 vs. 0.9804) —en otras palabras, identifica un número apenas mayor de animales—, lo hace a costa de más falsos positivos (113 vs. 88). En cambio, MC_C modera de forma significativa esas falsas detecciones y logra un rendimiento global superior en casi todas las métricas, evidenciando que el ajuste del umbral mediante la distancia (0,1) en la ROC permite un equilibrio más firme entre sensibilidad y precisión.

Por último, se ha presentado que MegaDetector también funciona con un umbral denominado conservador (0,05). Al observar las métricas de validación de esta configuración, se ha comprobado que, a medida que se aumentaba el valor del umbral, la métrica recall aumentaba su valor. Por tanto, se ha decidido buscar el umbral que haga que MegaClassifier obtenga el mismo recall.

Para obtener el umbral de MegaClassifier que iguale el recall de MegaDetector configurado de manera conservadora (umbral 0,05), se procedió evaluando las predicciones en el conjunto de validación a lo largo de un rango de umbrales y trazando la curva Recall vs. Umbral. Esta curva permitió observar cómo incrementaba la métrica recall a medida que se aumentaba el umbral, de forma similar a lo que ocurre en MegaDetector. A partir de dicha comparación, se identificó el valor de umbral en MegaClassifier que generaba un recall equivalente al de la configuración conservadora de MegaDetector, asegurando condiciones comparables para ambos modelos y posibilitando una evaluación justa de su capacidad de detección.

Tabla 7: Comparativa MegaDetector con MegaClassifier_C con umbrales conservadores

Modelo	Umbral	Accuracy	Loss	Precision	Recall	Specificity	F1-Score	AUC
MD	0.05	0,9326	-	0,9136	0,9919	0,8166	0,9511	-
MC_C	0.696	0,9375	0,0136	0,9090	0,9961	0,8481	0,9506	0,9937

Tras ajustar el umbral de MegaClassifier (MC_C) (ver resultados en Tabla 7) para igualar e incluso superar levemente el recall de la configuración conservadora de MegaDetector (MD, umbral = 0.05), se evidencia que MC_C alcanza un recall de 0.9961 frente a 0.9919 de MD, incrementando además la accuracy (0.9375 vs. 0.9326) y la specificity (0.8481 vs. 0.8166), aunque presenta una precision ligeramente inferior (0.9090 vs. 0.9136) y un F1-Score casi equivalente (0.9506 vs. 0.9511). Las matrices de confusión (ver Figura 27) refuerzan estos hallazgos, mostrando que, a

pesar de que MD comete un número marginalmente menor de falsos positivos, MC_C reduce significativamente los falsos negativos, lo que mejora la detección de la clase animal sin comprometer la identificación correcta de imágenes vacías. En conjunto, al optimizar el umbral para que el recall de MC_C se alinee con el de la versión conservadora de MD, se garantiza una alta sensibilidad y un desempeño global más robusto, consolidando a MC_C como una alternativa superior en términos de capacidad discriminativa.

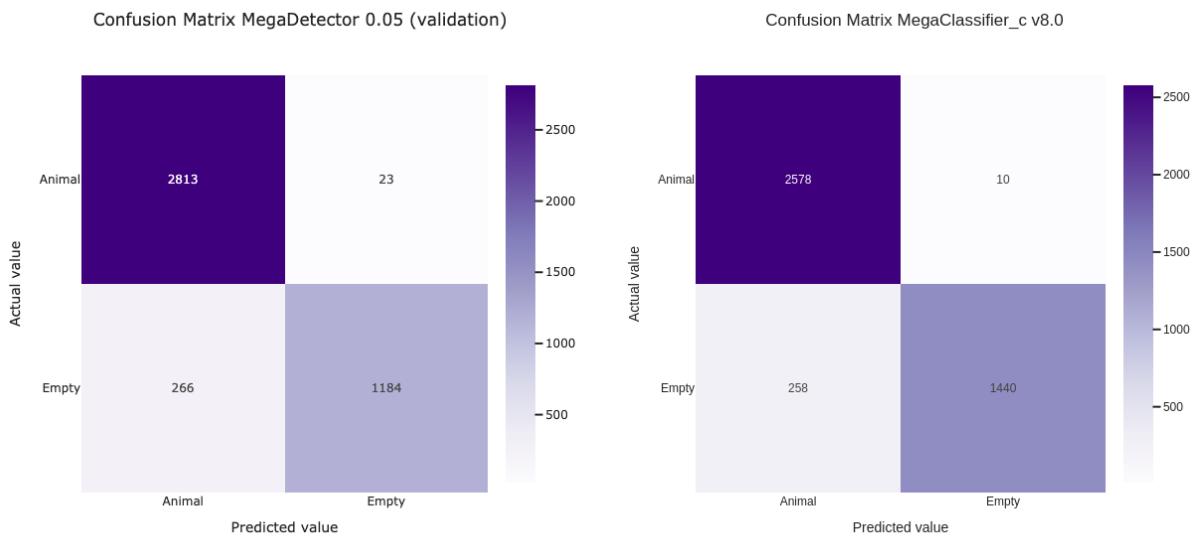


Figura 27: Comparativa de matrices de confusión de validación de MegaDetector y MegaClassifier_C con umbrales conservadores

Tabla 8: Resumen de métricas MegaClassifier y MegaDetector sobre conjunto de validación

Modelo	Umbral	Accuracy	Loss	Precision	Recall	Specificity	F1-Score	AUC
MD	0.2	0,9620	-	0,9610	0,9824	0,9221	0,9716	-
MD	0.1	0,9503	-	0,9400	0,9880	0,8766	0,9634	-
MD	0.05	0,9326	-	0,9136	0,9919	0,8166	0,9511	-
MC_C	0.5	0,9636	0,0136	0,9743	0,9708	0,9493	0,9725	0,9937
MC_C	0.5334	0,9666	0,0136	0,9690	0,9804	0,9407	0,9746	0,9937
MC_C	0.696	0,9375	0,0136	0,9090	0,9961	0,8481	0,9506	0,9937

Capítulo 5 Resultados y discusión

En el presente apartado se exponen los datos correspondientes a la ejecución de MegaDetector y MegaClassifier sobre el conjunto de prueba, en los diversos umbrales.

5.1 Resultados del conjunto de prueba

Como se ha explicado anteriormente, MegaDetector, en su documentación específica, recomienda ajustar el valor del umbral de confianza en función del contexto. En ausencia de especificaciones, se asume un umbral de 0,1. Por otro lado, para situaciones generales donde el modelo exhibe un desempeño equilibrado, se define el umbral típico, con un valor de 0,2. Por último, para aquellas situaciones en las que es imperativo detectar un mayor número de animales o en imágenes con condiciones más complejas, se recomienda el uso del umbral conservador, establecido en 0.05.

En consonancia con este criterio, durante la fase de entrenamiento se plantearon diversos valores de umbrales en MegaClassifier_C, con el propósito de que operaran de manera análoga a los valores previamente mencionados. En síntesis, se ha elaborado una tabla en la que se exhiben los valores del umbral por defecto (0.5), el umbral típico (0.5334) y el umbral conservador (0.679).

Tabla 9: Umbrales de los modelos

Umbrales	Valor en MegaDetector	Valor de MegaClassifier
Típico	0,2	0,5334
Por defecto	0,1	0,5
Conservador	0,05	0,696

En concordancia con lo expuesto en la sección precedente, se procederá a la presentación de los resultados obtenidos por cada umbral específico. En este sentido, se exhibirán los resultados del umbral conservador del clasificador en cuestión, en contraposición con los resultados del umbral conservador de MegaDetector, con el propósito de establecer una referencia más equitativa y así sucesivamente con el resto de los umbrales.

En lo que respecta al umbral previamente mencionado, los resultados obtenidos en el conjunto de prueba han sido los siguientes:

Tabla 10: Resultados del conjunto de prueba usando umbral conservador en MegaDetector y MegaClassifier_C

Modelo	Umbral	Accuracy	Loss	Precision	Recall	Specificity	F1-Score	AUC
MD	0,05	0,9288	-	0,9099	0,9905	0,8083	0,9485	-
MC_C	0,696	0,9391	0,0122	0,9104	0,9973	0,8503	0,9519	0,9947

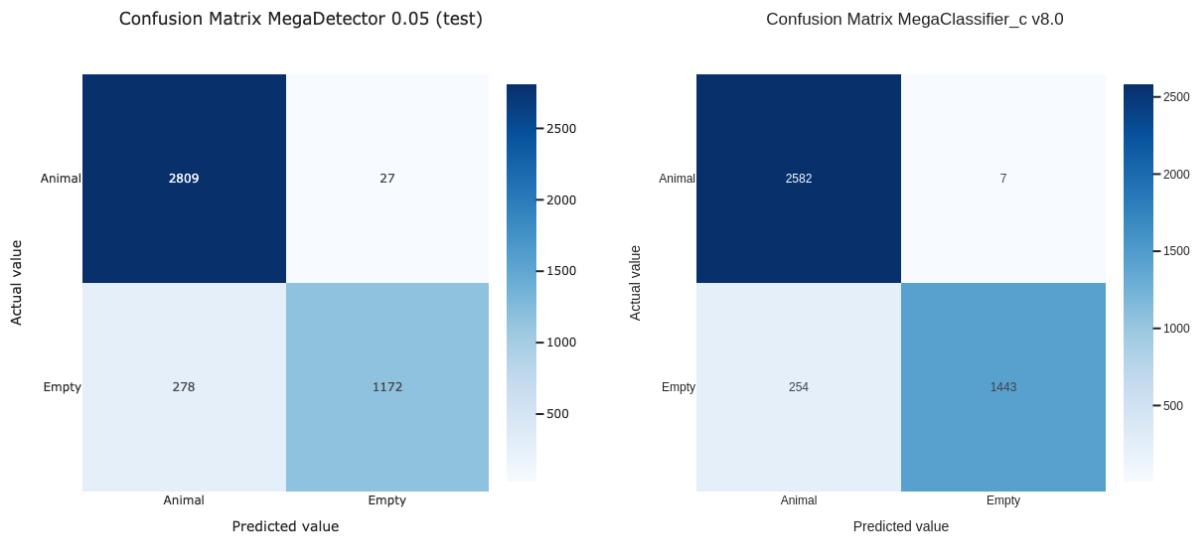


Figura 28: Comparativa de matrices de confusión de prueba de MegaDetector y MegaClassifier_C con umbrales conservadores

En la evaluación del conjunto de pruebas, MegaClassifier con el umbral ajustado (0,696) vuelve a superar al MegaDetector en casi todas las métricas (ver Tabla 10), incluso en su configuración conservadora (umbral = 0,05). Específicamente, MC_C aumenta la accuracy (0.9391 vs. 0.9288) y la precision (0.9104 vs. 0.9099), además de elevar significativamente el recall (0.9973 vs. 0.9905) y la specificity (0.8503 vs. 0.8083). El F1-Score también refleja dicha mejora (0.9519 vs. 0.9485), lo que evidencia un balance más robusto entre sensibilidad y precisión.

Las matrices de confusión corroboran estos hallazgos (ver Figura 28): mientras MD deja escapar 27 animales (falsos negativos), MC_C reduce esta cifra a 7, lo que explica su capacidad de recuperación superior. Además, MC_C equivoca 254 imágenes vacías como animales, frente a las 278 de MD, lo que también aumenta su especificidad. En consecuencia, la configuración de MC_C logra una cobertura extensa de la clase animal sin incurrir en un aumento significativo de los falsos positivos, consolidándose como una alternativa más efectiva que MegaDetector para la mayoría de las situaciones consideradas.

Como se introdujo en la sección anterior, durante la implementación de la CNN destinada a cubrir el objetivo principal del proyecto, se realizaron experimentos con el propósito de combinar MegaDetector y una CNN desarrollada con el mismo versionado que MegaClassifier_C, pero que utilizase como entrada los datos generados por la herramienta de Microsoft. En consecuencia, se presentan también los resultados de MegaClassifier_A y MegaClassifier_B.

Tabla 11: Resultados del conjunto de prueba usando umbral por defecto en MegaDetector y MegaClassifier_C

Modelo	Umbral	Accuracy	Loss	Precision	Recall	Specificity	F1-Score	AUC
MD	0,1	0,9482	-	0,9371	0,9880	0,8703	0,9619	-
MC_C	0,5	0,9641	0,0122	0,9714	0,9742	0,9444	0,9728	0,9947
MC_B	0,5	0,9608	0,0132	0,9556	0,9847	0,9179	0,9699	0,9940
MC_A	0,5	0,9622	0,0315	0,9654	0,9772	0,9340	0,9713	0,9939

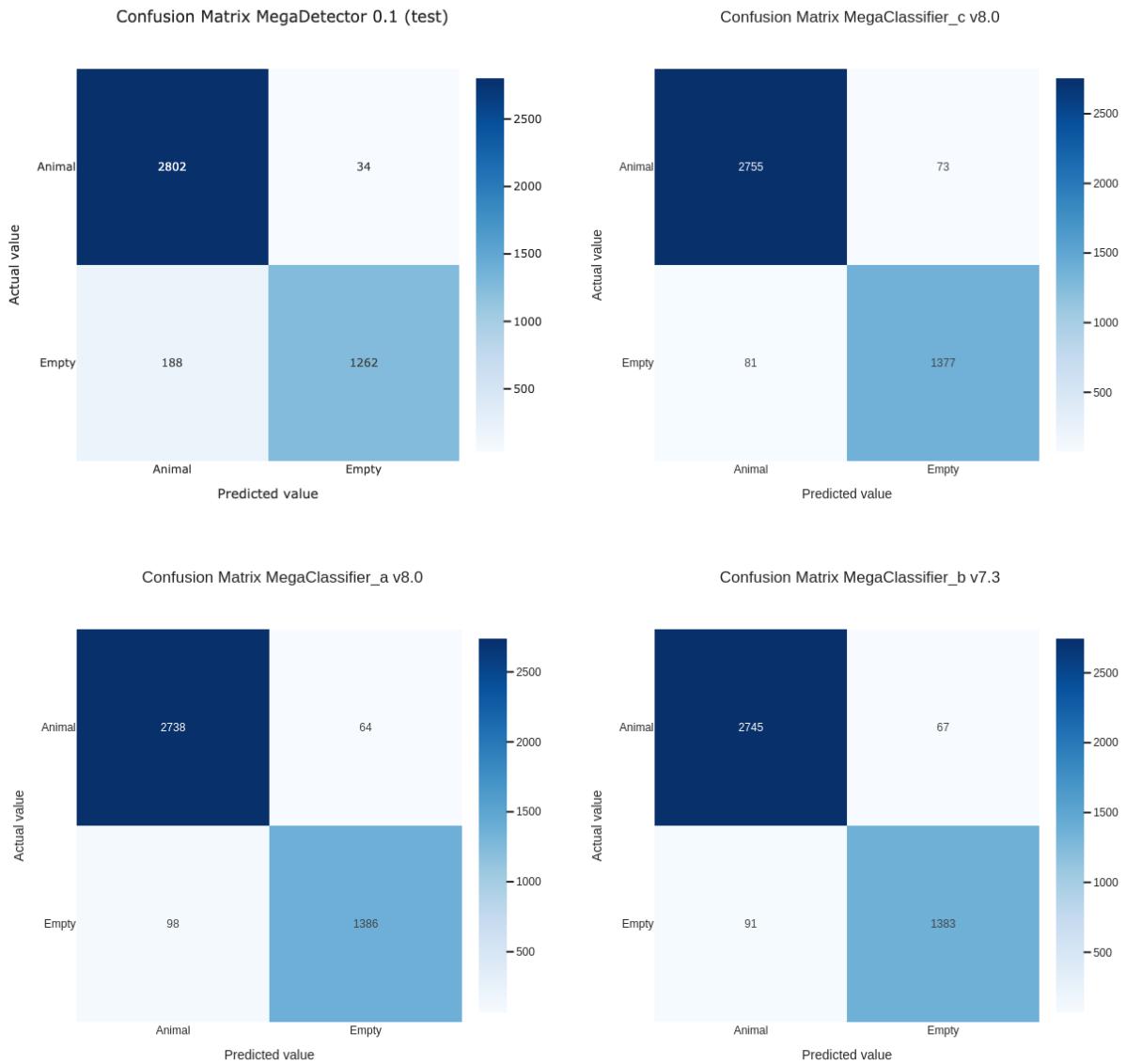


Figura 29: Comparativa de matrices de confusión de prueba de MegaDetector y MegaClassifier_C con umbrales por defecto

En la evaluación sobre el conjunto de prueba (ver Tabla 11 y Figura 29) se observa que MegaDetector (MD), utilizado como referencia, ofrece un rendimiento equilibrado en diversas métricas. Sin embargo, el modelo MC_C, que corresponde a la CNN de clasificación, se destaca

al lograr mejores resultados en términos de precisión, recall, F1-score y AUC, lo que evidencia una capacidad superior para discriminar entre imágenes con presencia o ausencia de animales. En contraste, la primera combinación implementada, MC_A, no aporta mejoras significativas: sus resultados son iguales o incluso inferiores a los de MD, lo que sugiere que esta integración no aprovecha adecuadamente el potencial de MegaDetector y, por tanto, no justifica su implementación. Por otro lado, MC_B, que también combina ambos enfoques, pero mediante un tratamiento de datos diferente, muestra una mejora moderada en algunas métricas; sin embargo, sus resultados no alcanzan el nivel de consistencia y solidez logrado por MC_C. En resumen, la comparativa en test confirma que, si bien se exploraron diversas estrategias de combinación, el desempeño óptimo y prometedor se concentra en MC_C, constituyendo la alternativa más favorable para aplicaciones reales.

Tabla 12: Resultados del conjunto de prueba usando umbral típico en MegaDetector y MegaClassifier_C

Modelo	Umbral	Accuracy	Loss	Precision	Recall	Specificity	F1-Score	AUC
MD	0,2	0,9608	-	0,9591	0,9827	0,9179	0,9707	-
MC_C	0,5334	0,9659	0,0122	0,9658	0,9824	0,9352	0,9740	0,9947

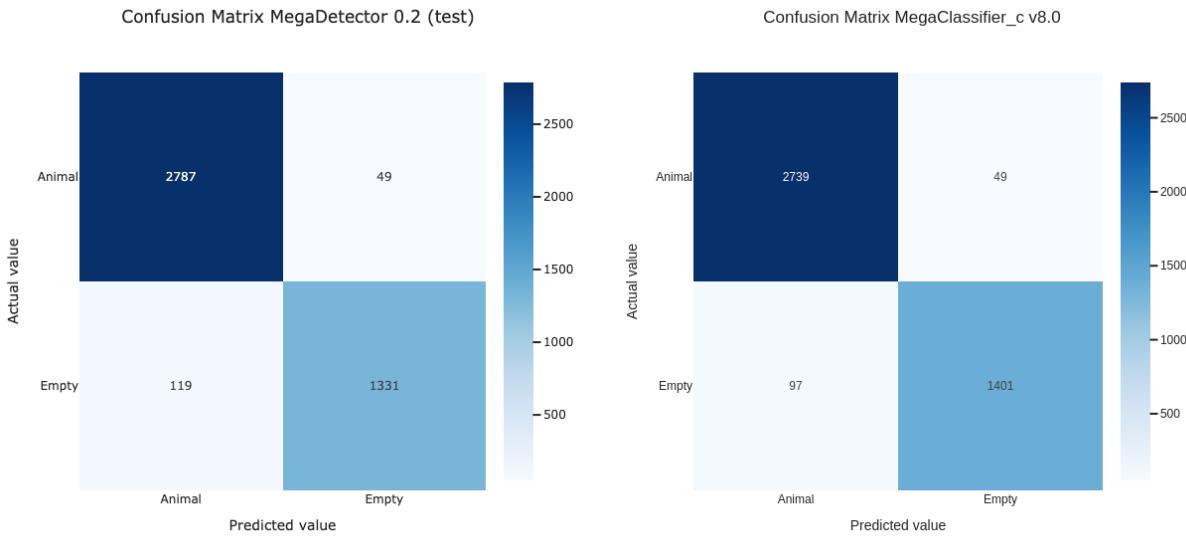


Figura 30: Comparativa de matrices de confusión de prueba de MegaDetector y MegaClassifier_C con umbrales típicos

Al comparar el rendimiento de MegaDetector (MD) con umbral 0.2 y MegaClassifier_C (MC_C) con umbral 0.5334 en el conjunto de prueba, se observa que MC_C ofrece mejoras notables en varias métricas clave. Aunque ambos modelos presentan un recall muy similar (0.9827 vs. 0.9824), MC_C muestra un incremento en la accuracy (0.9659 vs. 0.9608), en la specificity (0.9352 vs. 0.9179) y en el F1-Score (0.9740 vs. 0.9707).

Las matrices de confusión confirman estos resultados: aunque tanto MD como MC_C dejan escapar 49 animales (falsos negativos), MC_C reduce de 119 a 97 los falsos positivos (imágenes

vacías clasificadas erróneamente como animales). Esta combinación de alta sensibilidad y menor tasa de equivocaciones en la clase vacía otorga a MC_C un desempeño global más robusto que el de MD en umbral 0.2, respaldando su elección como mejor alternativa de clasificación para el escenario evaluado.

5.2 Análisis y discusión

Tras evaluar distintas configuraciones y umbrales para el modelo MegaClassifier_c (MC_C), se concluye que dicho clasificador ofrece un desempeño claramente superior al resto de alternativas, incluyendo la referencia establecida por MegaDetector. Además de mostrar valores sobresalientes en accuracy y F1-Score, MC_C alcanza un AUC cercano a 0.995, evidenciando su capacidad para separar de forma confiable las clases animal y vacío a lo largo de la curva ROC (ver Figura 31).

Tabla 13: Métricas del modelo final

Modelo	Umbral	Accuracy	Loss	Precision	Recall	Specificity	F1-Score	AUC
MC_C	0,5334	0,9659	0,0122	0,9658	0,9824	0,9352	0,9740	0,9947
MC_C	0,5	0,9641	0,0122	0,9714	0,9742	0,9444	0,9728	0,9947
MC_C	0,696	0,9391	0,0122	0,9104	0,9973	0,8503	0,9519	0,9947

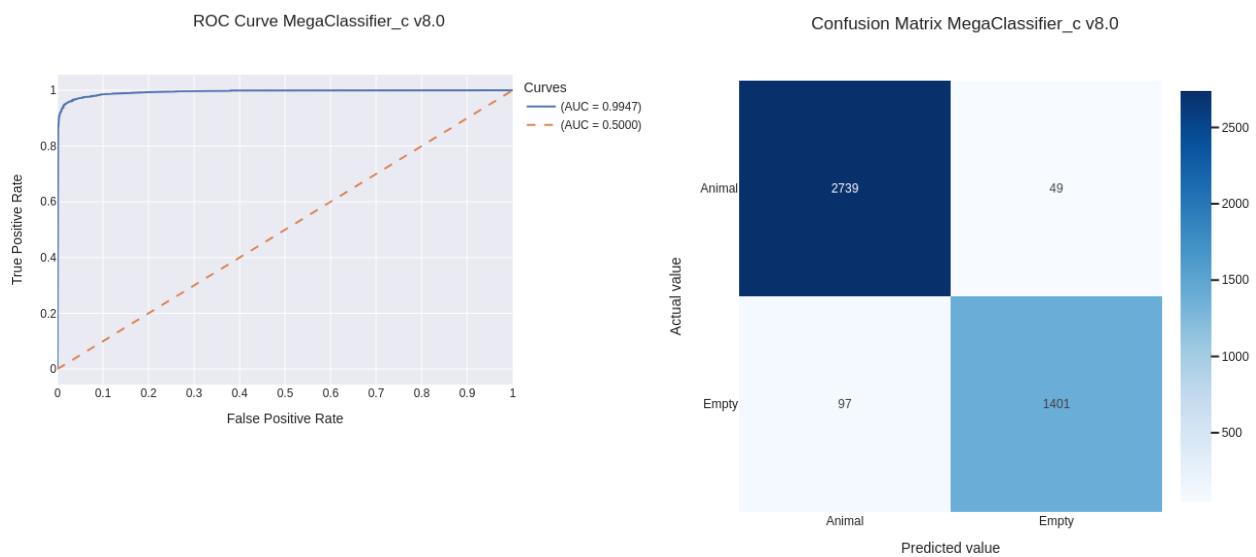


Figura 31: Curva ROC y matriz de confusión del modelo final sobre el conjunto de prueba

Uno de los aspectos clave en el perfeccionamiento de MC_C ha sido la determinación del umbral óptimo para la clasificación. Tras analizar la curva ROC en el conjunto de validación y confrontar los resultados obtenidos en test (ver Tabla 13), se identificó que un umbral cercano a 0.5334 ofrece un equilibrio particularmente eficiente entre sensibilidad y precisión. Con este valor:

- Se mantiene un recall (0.9824) lo suficientemente elevado para asegurar la detección de la mayoría de los animales.
- Se logra una precision competitiva (0.9658), minimizando la incidencia de falsos positivos.
- Se alcanza una specificity (0.9352) notable, reduciendo el número de imágenes vacías clasificadas erróneamente.
- El F1-Score (0.9740) se consolida como uno de los más altos entre todas las versiones consideradas.

Así, umbral = 0.5334 garantiza un excelente rendimiento global, tanto en el conjunto de validación como en el de prueba, lo que lo convierte en la opción más acertada para el despliegue del modelo. Esta elección supone un balance óptimo al priorizar la detección de animales sin comprometer en exceso la aparición de falsos positivos, favoreciendo al mismo tiempo la métrica de especificidad. De esta manera, MC_C se perfila como el mejor clasificador para el objetivo planteado, confirmando que la selección cuidadosa del umbral permite explotar al máximo sus capacidades discriminativas y robustecer su utilidad en aplicaciones reales.

Capítulo 6 Conclusiones y trabajos futuros

6.1 Conclusiones técnicas

A lo largo de este trabajo se ha desarrollado e implementado una red neuronal convolucional basada en transfer learning mediante la arquitectura EfficientNet-B5, cuyo objetivo principal ha sido la clasificación binaria de imágenes de fototrampeo en función de la presencia o ausencia de animales. Aunque MegaDetector, utilizado como modelo de referencia desarrollado por Microsoft, demuestra un rendimiento equilibrado en condiciones predeterminadas, la investigación ha permitido evidenciar que el modelo propuesto, MegaClassifier_C, supera significativamente a esta herramienta en múltiples métricas clave.

MegaClassifier_C ha alcanzado unos valores de rendimiento excepcionales, con una accuracy cercana al 96%, loss alrededor del 1%, precision del 96%, recall del 99%, specificity del 92% y un AUC de aproximadamente 0.995. Estos resultados demuestran que el clasificador es robusto y equilibrado, logrando una detección muy acertada de la presencia animal sin incurrir en un elevado número de falsos positivos. Un elemento fundamental en la mejora del rendimiento ha sido la optimización del umbral de decisión. Mediante el análisis de la curva ROC en el conjunto de validación, se determinó que un umbral de 0.5334 ofrece la mejor configuración para equilibrar la sensibilidad y la precisión del modelo. Esta elección ha permitido que MegaClassifier_C capture la mayoría de los casos positivos sin sacrificar la capacidad de discriminar correctamente entre imágenes con y sin animales.

Además, se han experimentado distintas estrategias de combinación con MegaDetector (MC_A y MC_B); sin embargo, estas variantes no han aportado mejoras significativas y, en algunos casos, han ofrecido resultados equivalentes o inferiores a los obtenidos con MegaClassifier_C. Esto reafirma la conveniencia de utilizar el modelo de clasificación puro, ajustado y optimizado, en lugar de recurrir a métodos híbridos que no explotan plenamente el potencial de cada componente.

En definitiva, la implementación y posterior optimización del modelo MegaClassifier_C ha demostrado ser una solución eficaz para la clasificación de imágenes de fototrampeo, posicionándose a la altura de herramientas de referencia desarrolladas por grandes compañías tecnológicas. La cuidadosa selección de hiperparámetros, la aplicación de técnicas de data augmentation, early stopping y fine tuning, junto con la optimización del umbral a 0.5334, han contribuido a consolidar un sistema con gran potencial de generalización en entornos reales, abriendo la puerta a futuras mejoras y aplicaciones en la monitorización automatizada de la fauna.

6.2 Trabajos futuros

A partir de los resultados obtenidos en este trabajo, se identifican varias líneas de desarrollo y mejora que podrían explorarse en trabajos futuros:

- **Evaluación en condiciones reales:** Sería útil aplicar el modelo en un entorno de monitoreo automatizado en campo, donde se puedan analizar aspectos como la velocidad de inferencia, robustez ante ruido y posibles falsos positivos/negativos en condiciones no controladas.
- **Optimización del modelo para dispositivos de bajo consumo:** Dado que uno de los posibles escenarios de uso es en estaciones de monitoreo en zonas remotas, podría estudiarse la posibilidad de comprimir el modelo sin comprometer significativamente el rendimiento.
- **Clasificación multicategoría o multietiqueta:** Una posible extensión del problema actual sería no solo detectar la presencia o ausencia de animales, sino también clasificar la especie o grupo taxonómico correspondiente, lo cual implicaría un modelo más complejo y la necesidad de un dataset más detallado.
- **Desarrollo de un modelo híbrido:** Se propone investigar la integración efectiva de MegaDetector con un clasificador CNN, ya sea profundizando en los experimentos planteados o explorando nuevos enfoques. Esto podría implicar la fusión de las salidas de ambos modelos a diferentes niveles (por ejemplo, combinando características o utilizando técnicas de ensemble) para aprovechar las fortalezas de cada uno. El objetivo sería desarrollar un sistema que mantenga la robustez y precisión de MegaDetector en la detección de animales, al tiempo que incorpora la capacidad discriminativa detallada del clasificador CNN, ofreciendo una solución más completa y adaptada a escenarios complejos.

6.3 Valoración personal

La realización de este Trabajo de Fin de Grado ha supuesto una experiencia enriquecedora tanto a nivel académico como personal. A lo largo del desarrollo del proyecto, se ha podido profundizar en el campo del aprendizaje profundo, en particular en la construcción y evaluación de modelos de clasificación de imágenes mediante redes neuronales convolucionales y técnicas de transfer learning.

Durante el desarrollo del trabajo también he podido fortalecer habilidades clave como la gestión del tiempo, la toma de decisiones técnicas basadas en resultados empíricos, y la capacidad de análisis crítico. Asimismo, he ganado confianza en el uso de herramientas como TensorFlow, Keras, y técnicas de visualización y análisis de métricas.

En resumen, este proyecto no solo ha sido una oportunidad para aplicar conocimientos técnicos, sino también un proceso de aprendizaje completo que me ha motivado a seguir profundizando en el campo del aprendizaje automático y su aplicación en entornos reales.

Referencias

- [1] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet classification with deep convolutional neural networks,» 2012. [En línea]. Available: <https://doi.org/10.1145/3065386>.
- [2] K. Simonyan y A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition,» 2014. [En línea]. Available: <https://doi.org/10.48550/arXiv.1409.1556>.
- [3] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» 2015. [En línea]. Available: <https://doi.org/10.1109/CVPR.2016.90>.
- [4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed y D. Anguelov, «Going deeper with convolutions,» 2015. [En línea]. Available: <https://doi.org/10.1109/CVPR.2015.7298594>.
- [5] M. Tan y Q. V. Le, «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,» 2019. [En línea]. Available: <https://doi.org/10.48550/arXiv.1905.11946>.
- [6] J. Yosinski, J. Clune, Y. Bengio y H. Lipson, «How transferable are features in deep neural networks?,» 2014. [En línea]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/532a2f85b6977104bc93f8580abbb330-Paper.pdf.
- [7] M. S. Norouzzadeh, A. Nguyen, M. Kosmala, A. Swanson, M. S. Palmer, C. Packer y J. Clune, «Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning,» 2018. [En línea]. Available: <https://doi.org/10.1073/pnas.1719367115>.
- [8] M. A. Tabak, M. S. Norouzzadeh, D. W. Wolfson, S. J. Sweeney, K. C. Vercauteren, N. P. Snow, J. M. Halseth, P. A. D. Salvo, J. S. Lewis, M. D. White, B. Teton, J. C. Beasley, P. E. Schlichting y R. K. Boughto, «Machine learning to classify animal species in camera trap images: Applications in ecology,» 2019. [En línea]. Available: <https://doi.org/10.1111/2041-210X.13120>.
- [9] D. Morris, S. Beery, N. Joshi y N. Jovic, «MegaDetector: Object Detection for Camera Trap Images. Microsoft AI for Earth,» [En línea]. Available: <https://github.com/microsoft/CameraTraps>.
- [10] L. N. Smith, «Cyclical Learning Rates for Training Neural Networks,» 2017. [En línea]. Available: <https://doi.org/10.1109/WACV.2017.58>.

- [11] M. Buda, A. Maki y M. A. Mazurowski, «A systematic study of the class imbalance problem in convolutional neural networks,» 2018. [En línea]. Available: <https://doi.org/10.1016/j.neunet.2018.07.011>.
- [12] A. Turing, «On Computable Numbers, with an Application to the Entscheidungsproblem,» 1936. [En línea]. Available: <https://doi.org/10.1112/plms/s2-42.1.230>.
- [13] W. S. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity,» 1943. [En línea]. Available: <https://doi.org/10.1007/BF02478259>.
- [14] A. Turing, «Computing Machinery and Intelligence,» 1950. [En línea]. Available: <https://doi.org/10.1093/mind/LIX.236.433>.
- [15] F. Rosenblatt, «The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,» 1958. [En línea]. Available: <https://doi.org/10.1037/h0042519>.
- [16] B. Widrow, «An Adaptive "ADALINE" Neuron Using Chemical "Memistors",» 1960. [En línea]. Available: <https://isl.stanford.edu/~widrow/papers/t1960anadaptive.pdf>.
- [17] M. Minsky y S. A. Papert, «Perceptrons: An Introduction to Computational Geometry,» 1969. [En línea]. Available: <https://doi.org/10.7551/mitpress/11301.001.0001>.
- [18] D. E. Rumelhart, G. E. Hilton y R. J. Williams, «Learning representations by back-propagating errors,» 1986. [En línea]. Available: <https://doi.org/10.1038/323533a0>.
- [19] K. Hornik, M. Stinchcombe y H. White, «Multilayer feedforward networks are universal approximators,» 1989. [En línea]. Available: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8).
- [20] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard y L. D. Jackel, «Backpropagation Applied to Handwritten Zip Code Recognition,» 1989. [En línea]. Available: <https://doi.org/10.1162/neco.1989.1.4.541>.
- [21] J. Hochreiter, «Untersuchungen zu dynamischen neuronalen Netzen,» 1991. [En línea]. Available: <https://www.bioinf.jku.at/publications/older/3804.pdf>.
- [22] G. E. Hinton, S. Osindero y Y.-W. Teh, «A Fast Learning Algorithm for Deep Belief Nets,» 2006. [En línea]. Available: <https://doi.org/10.1162/neco.2006.18.7.1527>.
- [23] G. E. Hinton, «Training Products of Experts by Minimizing Contrastive Divergence,» 2002. [En línea]. Available: <https://doi.org/10.1162/089976602760128018>.

- [24] H. Robbins y S. Monro, «A Stochastic Approximation Method,» 1951. [En línea]. Available: <https://doi.org/10.1214/aoms/1177729586>.
- [25] B. Polyak, «Some methods of speeding up the convergence of iteration methods,» 1964. [En línea]. Available: [https://doi.org/10.1016/0041-5553\(64\)90137-5](https://doi.org/10.1016/0041-5553(64)90137-5).
- [26] D. P. Kingma y J. Ba, «Adam: A Method for Stochastic Optimization,» 2015. [En línea]. Available: <https://doi.org/10.48550/arXiv.1412.6980>.
- [27] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection,» 2015. [En línea]. Available: <https://doi.org/10.48550/arXiv.1506.02640>.
- [28] X. G. y Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, 2008.
- [29] K. He, X. Zhang, S. Ren y J. Sun, «Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,» 2015. [En línea]. Available: <https://doi.org/10.1109/ICCV.2015.123>.

Bibliografía

- [TensorFlow] An end-to-end platform for machine learning. [ENLACE](#)
- [Keras] A superpower for ML developers. [ENLACE](#)
- [Python] Welcome to Python.org. [ENLACE](#)
- [Visual Studio Code] Your code editor. Redefined with AI. [ENLACE](#)
- [Anaconda] The Operating System for AI. [ENLACE](#)
- [EfficientNet] EfficientNet B0 to B7 . [ENLACE](#)
- [Microsoft] MegaDetector. [ENLACE](#)
- [Microsoft AI for Earth] Repositorio general de CameraTraps. [ENLACE](#)
- [Cookiecutter Data Science] Plantilla proyecto repositorio. [ENLACE](#)
- [Youtube] Food Image Classification [ENLACE](#)
- [Youtube] Deep Learning Tutorial 5 - Multiclass Flowers Classification using VGG16 with Transfer Learning. [ENLACE](#)
- [Youtube] Efficient Image Classification with Transfer Learning and Image Augmentation with TensorFlow Keras. [ENLACE](#)
- [Youtube] Build a Deep CNN Image Classifier with ANY Images. [ENLACE](#)
- [Youtube] Transfer Learning with Keras and TensorFlow: How to Fine-Tune a Pretrained Neural Network. [ENLACE](#)
- [Youtube] Crea un clasificador de perros y gatos con IA, Python y Tensorflow - Proyecto completo. [ENLACE](#)
- [Youtube] ¿Pocos datos de entrenamiento? Prueba esta técnica. [ENLACE](#)
- [Pytorch Lightning] Learning rate Finder. [ENLACE](#)
- [AnalyticsVidhya] What is Machine Learning? [ENLACE](#)
- [Towards Data Science] A tour of Machine Learning Algorithms. [ENLACE](#)
- La neurona artificial de McCulloch -Pitts. [ENLACE](#)
- [Medium] Tipos de aprendizaje automático. [ENLACE](#)
- [AdeShpand33] A Beginner's Guide To Understanding Convolutional Neural Networks [ENLACE](#)
- [AdeShpande3] Why Machine Learning Is a Metaphor for Life [ENLACE](#)
- [Udacity] Intro to Tensorflow for Deep Learning [ENLACE](#)
- [SpringerOpen] Convolutional neural networks: an overview and application in radiology [ENLACE](#)
- [TechTarget] A timeline of Machine Learning History. [ENLACE](#)
- [Data Topics] A Brief History of Machine Learning. [ENLACE](#)
- [Wikipedia] Restricted Boltzmann machine. [ENLACE](#)
- [Towards Data Science] AlexNet : The Architecture that Challenged CNNs [ENLACE](#)
- [ProgrammerClick] Use la red neuronal convolucional (CNN) para el reconocimiento facial [ENLACE](#)
- [Towards Data Science] Transposed Convolution Demystified. [ENLACE](#)
- [Medium] Unsampling: Unpooling and Transpose Convolution. [ENLACE](#)
- Introduction to VGG16 [ENLACE](#)
- [Viso] Very Deep Convolutional Networks (VGGNet) [ENLACE](#)
- [Towards Data Science] Deep Learning: GoogleNet Explained [ENLACE](#)
- [GeekForGeeks] Understanding GoogleNet Model [ENLACE](#)

[Medium] Inception V1 Architecture [ENLACE](#)

[Towards Data Science] Cross-Entropy Loss Function [ENLACE](#)

Loss Functions [ENLACE](#)

[Wikipedia] Mean Squared Error [ENLACE](#)

[Machine Learning Mastery] Dropout for Regularizing Deep Neural Networks [ENLACE](#)

ANEXOS

Anexo teórico A Introducción e historia del Deep Learning

Al abordar la resolución de una tarea mediante el uso de un ordenador, es imprescindible proporcionar una secuencia ordenada de pasos que el sistema debe seguir para lograr su correcta ejecución, lo que se conoce comúnmente como algoritmo. Identificar estos pasos no siempre es fácil, sobre todo cuando nos enfrentamos a tareas muy complejas, como las relacionadas con la visión artificial, la conducción autónoma o el reconocimiento de voz.

Por consiguiente, debemos determinar meticulosamente los pasos que nuestro algoritmo debe seguir. Este proceso implica especificar con antelación las instrucciones que el sistema informático deberá seguir para completar la tarea deseada. Los algoritmos se introducen en el ordenador en forma de programas escritos en algún lenguaje de programación, lo que permite que la máquina ejecute las operaciones necesarias para alcanzar el objetivo deseado.

A partir de este planteamiento, se extrae que para realizar cualquier tarea con un ordenador primero tenemos que dar la secuencia de pasos correcta que, una vez ejecutada, nos lleve a su solución. Sin embargo, cuando nos enfrentamos a problemas más complejos, elaborar un algoritmo capaz de resolverlos se convierte en todo un reto para el desarrollador.

Este es el punto de partida de los algoritmos de Deep Learning; sus técnicas tienen como fin que el propio sistema que implementamos encuentre esos pasos para conseguir la resolución del problema.

A.2 Introducción al Deep Learning

El Deep Learning se inspira en el proceso de aprendizaje humano, un enfoque bioinspirado que se basa en nuestra capacidad innata para adquirir y perfeccionar habilidades a lo largo de la vida. Los seres humanos interiorizamos diversas tareas mediante un proceso de práctica y repetición, como aprender a hablar, caminar o conducir. De manera similar, los algoritmos de Machine Learning analizan datos y patrones para mejorar su rendimiento y ofrecer soluciones eficientes a problemas complejos.

Como hemos mencionado, el conjunto de algoritmos aprende a resolver problemas mediante el análisis de datos. Durante el proceso de entrenamiento, se alimenta a un algoritmo con numerosos ejemplos que representan el problema. Así, el algoritmo detecta patrones y relaciones en los datos sin necesidad de recibir instrucciones específicas sobre los datos que debe seguir.

El desarrollo de un sistema de Machine Learning consta de varias etapas clave. En primer lugar, el modelo se entrena con datos específicos y se perfecciona mediante la práctica. A continuación, en la fase de validación, el modelo se evalúa con nuevos datos para comprobar su

efectividad, de manera similar a como los humanos se someten a exámenes para medir sus conocimientos.

Como se observa en la Figura 32, todo proceso completo para obtener un modelo funcional consta de más etapas que las mencionadas anteriormente. A continuación, describimos y explicamos los objetivos de cada una de ellas:

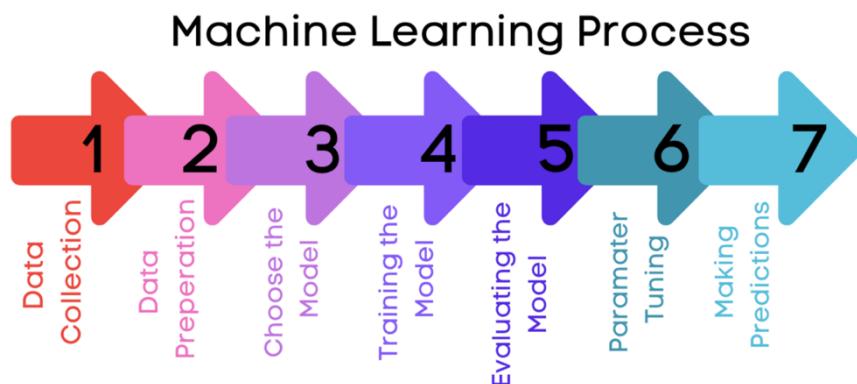


Figura 32: Etapas del proceso de Machine Learning¹⁴

- **Etapa 1:** Recopilación de datos. Una vez definido el problema que hay que resolver, el primer paso será recopilar datos relevantes. Esto implica obtener datos de diferentes fuentes, como bases de datos, sensores, API o archivos. El objetivo es asegurarse de tener la información suficiente para entrenar un modelo robusto y representativo.
- **Etapa 2:** Preparación de los datos. Con los datos recopilados, se procede a tratarlos y limpiarlos para que estén listos para entrenar el modelo. Esto puede incluir la eliminación de valores faltantes, la corrección de datos erróneos, la normalización o estandarización de los datos y la transformación de variables categóricas en numéricas. El objetivo es sencillo: obtener un conjunto de datos limpio y estructurado que puede utilizarse eficazmente por los algoritmos.
- **Etapa 3:** Selección del modelo. Dentro del Machine Learning existe una gran variedad de algoritmos disponibles. La elección de uno u otro puede depender de factores como la naturaleza del problema, el tamaño del conjunto de datos y sus características específicas.
- **Etapa 4:** Entrenamiento. Como mencionamos anteriormente, en esta fase se utiliza el conjunto de datos de entrenamiento para ajustar los parámetros del modelo.
- **Etapa 5:** Evaluación. Una vez que se considera el modelo entrenado, se evalúa su rendimiento utilizando un conjunto de datos de validación o prueba. Para ello, se utilizan métricas de evaluación apropiadas para medir la precisión, la exactitud, etc. El objetivo es determinar qué tan bien se está desempeñando el modelo y si es necesario realizar ajustes.
- **Etapa 6: Ajuste de parámetros.** Llegados al punto de obtener un modelo que haya aprendido correctamente, podemos seguir realizando pruebas para optimizar los

¹⁴ <https://redmond.ai/wp-content/uploads/2023/05/word-image-583-2.png>

hiperparámetros del modelo y mejorar así su rendimiento. Una posible forma de proceder sería fijar un parámetro, por ejemplo, el tamaño de la red; y realizar varios entrenamientos con diferentes tasas de aprendizaje para poder comparar cuál de ellos presenta un mejor comportamiento.

- **Etapa 7:** Predicción. Cuando consideramos que nuestro modelo está preparado, lo evaluaremos con el dataset de prueba. De esta forma, podremos estimar cuál será su rendimiento futuro y si es adecuado para resolver la tarea inicialmente planteada.

Estas son las características más habituales en la mayoría de los desarrollos de Machine Learning. Dado que este campo posee una gran variedad de algoritmos con diferentes propósitos, pueden existir diferentes clasificaciones.

Según la forma que se representa el conocimiento aprendido del modelo, se distinguen tres tipos:

- **Aprendizaje numérico:** Se basa en la obtención de conocimiento representado por valores numéricos, como los pesos en una red neuronal, sin que exista una relación directa con conceptos específicos. Ejemplos de este enfoque son las CNN, utilizadas en este trabajo, y las SVM¹⁵.
- **Aprendizaje simbólico:** En este caso, se centra en adquirir conocimientos que permiten representar conceptos mediante valores de atributos y reglas lógicas. Un ejemplo de este enfoque, utilizado para construir árboles de decisión es el algoritmo ID3¹⁶.
- **Aprendizaje mixto:** Combina elementos del aprendizaje numérico y simbólico, lo que permite la adquisición de conceptos a través de relaciones entre valores y atributos. Un ejemplo de este enfoque es XGBoost¹⁷.

También podemos clasificar según la información proporcionada durante el entrenamiento:

- **Aprendizaje supervisado:** Este tipo se caracteriza por tener salidas conocidas para cada entrada, lo que permite ajustar los valores internos del algoritmo y aproximarse a los resultados correctos.
- **Aprendizaje no supervisado:** A diferencia del anterior, este tipo de aprendizaje opera sin salidas conocidas, por lo que se enfoca en la identificación de patrones en los datos de entrada.
- **Aprendizaje por refuerzo:** Tampoco se conocen las salidas, pero la diferencia estriba en la utilización de un sistema de recompensas y castigos para guiar al modelo hacia la

¹⁵ Support Vector Machine, algoritmo de clasificación que encuentra el hiperplano óptimo para separar distintas clases en un espacio de alta dimensión.

¹⁶ Iterative Dichotomiser 3, basado en árboles de decisión que construye el modelo seleccionando el atributo más informativo en cada división.

¹⁷ Extreme Gradient Boosting, algoritmo basado en árboles de decisión optimizados mediante boosting, diseñado para ser eficiente y preciso en tareas de clasificación y regresión.

optimización de sus resultados. Esto hace que vaya aprendiendo en función de las acciones tomadas en distintos estados.

Podríamos realizar una clasificación en función del tipo de problema que se pretende solucionar. Algunos de los problemas más comunes son:

- **Regresión:** Su objetivo es modelar la relación entre los atributos de entrada y la salida del sistema de naturaleza numérica o cuantitativa.
- **Clasificación:** Su objetivo es modelar la relación entre la entrada y la salida cualitativa o categórica del sistema.

A continuación, hablaremos sobre algunos de los algoritmos utilizados en el Machine Learning para resolver los problemas mencionados en la Figura 33.

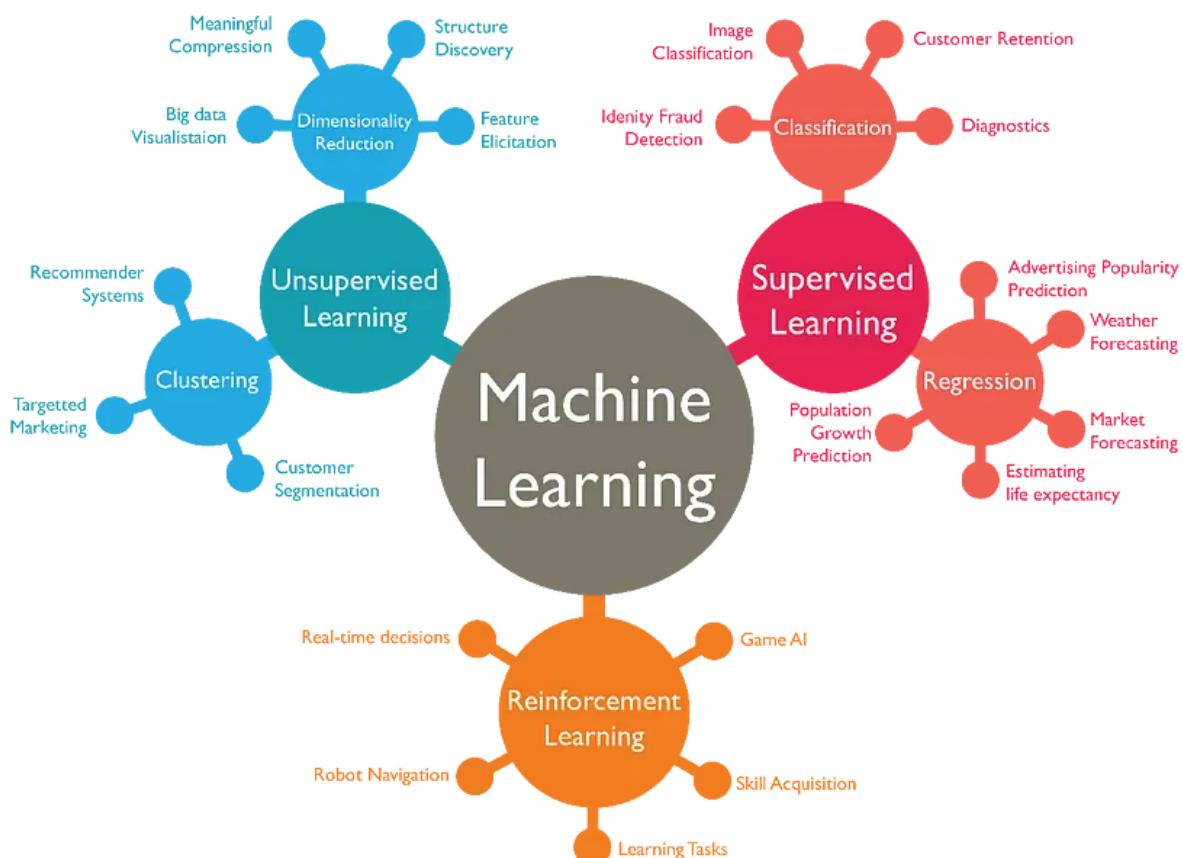


Figura 33: Esquema de ejemplos de aplicaciones del Machine Learning¹⁸

- **Algoritmos de regresión:** Son esenciales para predecir valores y hacer uso de la métrica del error para minimizarlo en cada iteración. Algunos ejemplos son: Regresión lineal, descenso por gradiente, etc.

¹⁸ <https://media.linkedin.com/dms/image/D5612AQHT5uzEz5mZ8g/...>

- **Algoritmos de reducción de dimensión:** Se aplican en aprendizaje no supervisado y su peculiaridad es que permiten reducir las características de un modelo complejo para facilitar su interpretación. Algunos ejemplos de este tipo de algoritmos son Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), etc.
- **Algoritmos Bayesianos:** Se utilizan en problemas de clasificación y regresión y se basan en teoremas de probabilidad, especialmente en el teorema de Bayes. Algunos ejemplos son: Bayesian Network, Naive Bayes, etc.
- **Máquina vector soporte:** Se ha mencionado en apartados anteriores. Conocida originalmente como Support Vector Machine, crea modelos que pueden mapear datos a un espacio de mayor dimensión, lo que permite encontrar un hiperplano que separe las clases y maximice el margen entre ellas.
- **Algoritmos de Clustering:** Se utilizan principalmente para agrupar datos de los que desconocemos sus características en común. Por este motivo, se utilizan en el aprendizaje no supervisado. Un ejemplo muy conocido es K-nearest Neighbors (KNN).
- **Algoritmos de Redes neuronales:** Se basan en el funcionamiento del cerebro humano. Suelen utilizarse para la clasificación y la regresión, aunque tienen un gran potencial para resolver problemas variados y son la base del Deep Learning. Los ejemplos más habituales son: Perceptrón, perceptrón multicapa, backpropagation, etc.
- **Algoritmos de Deep Learning:** Han surgido como la evolución de las redes neuronales anteriormente mencionadas. Su gran crecimiento se debe al abaratamiento de la tecnología actual, a la reducción de los costes computacionales y a la gran cantidad de datos de los que disponemos hoy en día. Algunos ejemplos de este tipo de algoritmos son: Convolutional Neural Network (CNN), Hierarchical Convolutional Deep Maxout Networks (HCDMN), Long Short Term Memory Neural Networks (LSTMNN), etc.

Un aspecto en la metodología de la gran mayoría de los algoritmos de Machine Learning es la correcta selección de los datos que tomaremos como representativos para la entrada. La correcta formación del modelo está estrechamente vinculada a la forma en que se representan los datos previamente.

Como vimos en la Figura 32, en la primera etapa del proceso, que consiste en la obtención de datos, es importante especificar las características que mejor definen cada una de las instancias del problema que se va a tratar. La correcta selección de estas características favorecerá que el aprendizaje alcance un nivel óptimo de calidad en los resultados. En caso contrario, si no se definen correctamente estas características, puede ser necesario utilizar un mayor número de ellas y, por tanto, un modelo más complejo, que en el peor de los casos puede impedir que este alcance un resultado aceptable.

Una de las maneras más generales de seleccionar estas características es utilizar aquellas que estadísticamente mejor definen el problema. Otra metodología consiste en consultar a expertos en la materia, ya que gracias a su experiencia pueden proporcionar información valiosa y relevante para realizar una selección correcta.

Llegados a este punto, nos damos cuenta de una de las grandes diferencias entre los algoritmos *convencionales* de Machine Learning y las nuevas técnicas de Deep Learning. En los primeros, hay que indicar previamente las características más relevantes, mientras que en las técnicas de Deep Learning es el propio algoritmo quien las encuentra por sí mismo durante la fase de entrenamiento. (ver Figura 34).

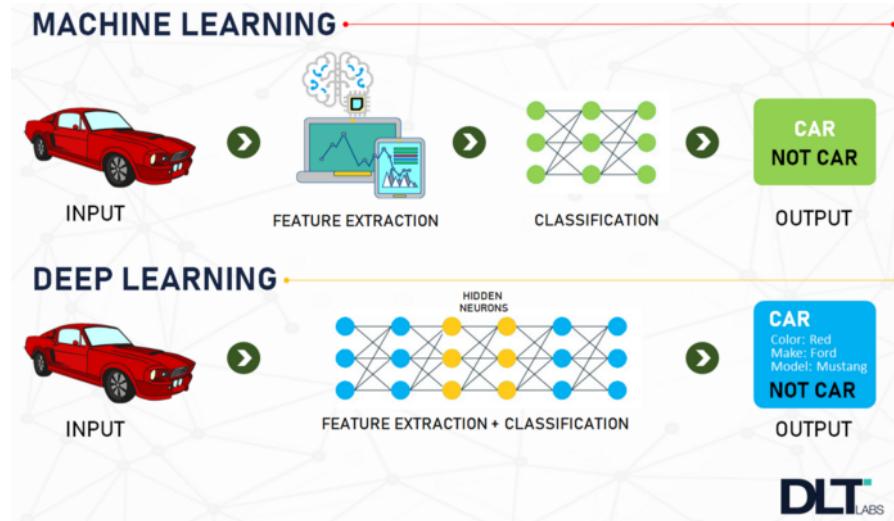


Figura 34: Comparativa entre Machine Learning y Deep Learning¹⁹

A.3 Historia del Deep Learning

En el anterior apartado se exponían los fundamentos del Deep Learning, describiendo su funcionamiento y las distintas etapas que conforman el desarrollo de un modelo basado en este paradigma. A partir de estos conceptos, en este apartado analizaremos la evolución histórica de las técnicas de Machine Learning, desde sus primeras aproximaciones hasta los avances más recientes que han permitido consolidar el aprendizaje profundo como una herramienta clave en múltiples ámbitos tecnológicos.

Al hablar de la historia del Deep Learning, también hay que abordar la historia del Machine Learning, ya que, como hemos comentado anteriormente, fue el que sentó sus bases. A lo largo de su evolución, el Machine Learning ha pasado por diferentes etapas, cada una de ellas marcada por el avance en nuevas técnicas y aplicaciones.

¹⁹ [https://media.linkedin.com/dms/image/D4D12AQH1_eSvKLwTiA/\[...\]](https://media.linkedin.com/dms/image/D4D12AQH1_eSvKLwTiA/[...])

Aunque el actual auge del Machine Learning nos pueda llevar a confusión, su desarrollo no es reciente, sino que se remonta a hace muchos años (ver Figura 35) , cuando no era fácil y su camino fue largo y complejo. Podríamos dividir su evolución en tres etapas clave, algunas de las cuales suscitan menos interés debido a las limitaciones de los modelos existentes en aquel momento. Estos altibajos son los que han provocado que en la actualidad se encuentre tan avanzado y diversificado, y sea una de las herramientas más poderosas, capaz de resolver problemas complejos, como el procesamiento de imágenes, que abordaremos en este trabajo de fin de grado.

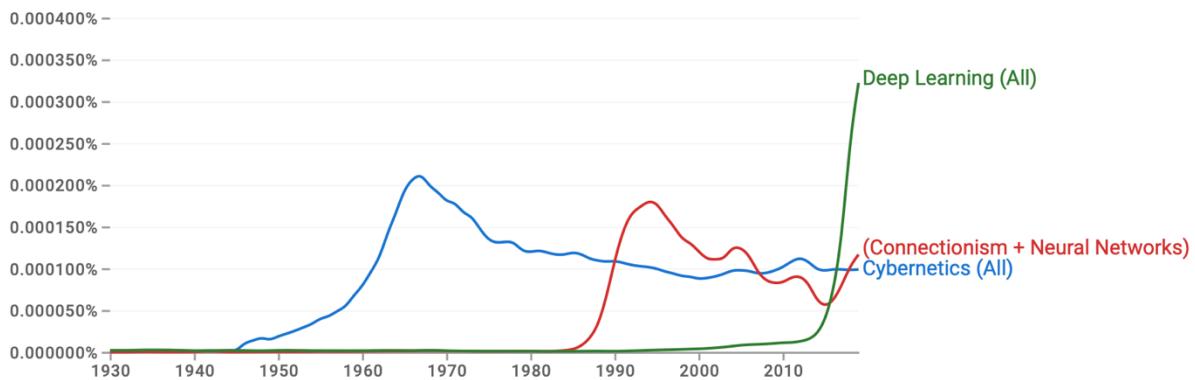


Figura 35: Frecuencia de aparición de conceptos relacionados con el Machine Learning en publicaciones a lo largo de la historia²⁰

A.3.1 Etapa Cibernetica (1940-1960)

En 1936, Alan Turing publicó un estudio en el que describió las máquinas de Turing [1], formalizando el concepto de algoritmo, lo que marcó el comienzo de la informática moderna.

En la década de 1940, Warren McCulloch y Walter Pitts presentaron un modelo matemático inspirado en las neuronas biológicas en 1943 [2]. Este modelo consistía en recibir un conjunto de entradas n ($x_1, x_2, x_3, \dots, x_n$) que se multiplicaban por pesos asociados w ($w_1, w_2, w_3, \dots, w_n$) y se sumaban. La salida dependía de una función de activación umbral que devolvía 1 si el valor era positivo y 0 si era negativo. Con un ajuste adecuado de los pesos, esta neurona podía realizar operaciones lógicas simples, como AND²¹, OR²² y NOT²³, lo que permitió usarla para razonamientos lógicos sencillos.

En 1950, Alan Turing publicó un artículo en el que formulaba una pregunta fundamental que revolucionó el campo de la computación: ¿Puede una máquina pensar?. En este artículo [3], Turing presentó su famoso *Test de Turing*. Gracias a este artículo, Alan Turing es considerado el padre de la informática.

²⁰ [https://books.google.com/ngrams/\[...\]](https://books.google.com/ngrams/[...])

²¹ Operación lógica de disyunción.

²² Operación lógica de conjunción.

²³ Operación lógica de negación.

Posteriormente, en 1958, Frank Rosenblatt propone el modelo del perceptrón, basándose en el modelo matemático de Walter Pitts y Warren McCulloch [4]. Se trataba del primer modelo matemático capaz de *aprender* a representar una función ajustando los pesos de sus entradas a partir de ejemplos dados.

Posteriormente, comenzaron a surgir modelos de redes neuronales que combinaban varios perceptrones, lo que permitía clasificar más de dos clases y asignar cada una a un perceptrón. En 1960, de la mano de Bernard Widrow, nació el modelo ADELINe [5], que tenía la peculiaridad de permitir cuantificar el error y ajustar los pesos durante el entrenamiento en función del error cometido.

Pese a todo ello, los modelos seguían siendo lineales. En 1969, Marvin Minsky y Seymour Papert demostraron que los perceptrones no eran capaces de resolver funciones no lineales, como la función lógica XOR, también llamada OR exclusive [6]. (ver Figura 36).

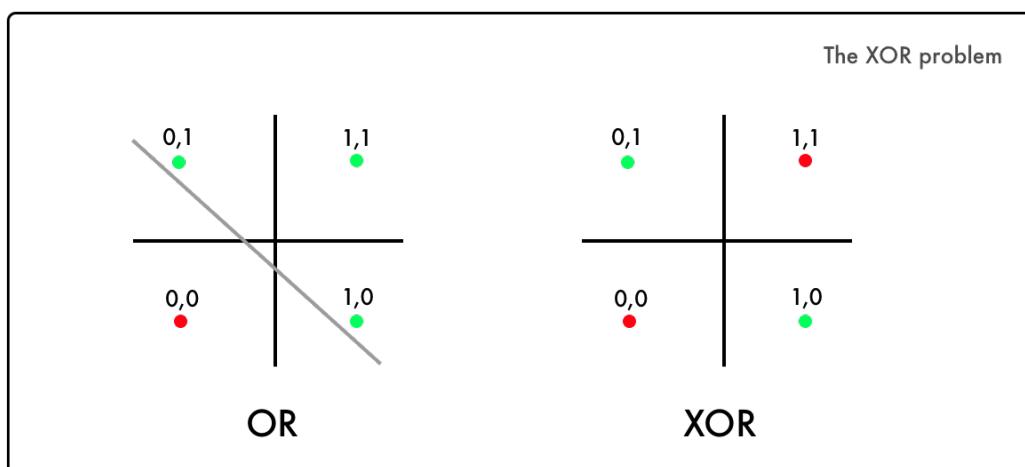


Figura 36: Demostración de cómo la función lógica XOR no puede modelarse linealmente²⁴

²⁴ <https://albertotb.com/curso-ml-R/Rmd/12-nn/img/xor.png>

A.3.2 Etapa Conexionismo (1980 – 1995)

A pesar del desinterés, las investigaciones en el campo del Machine Learning no se interrumpieron por completo. En 1986, David Rumelhart, Geoffrey Hinton y Ronald Williams redescubrieron el algoritmo de retropropagación, también conocido como backpropagation [7]. Gracias a este algoritmo, las redes multicapa podían entrenar eficazmente y modelar funciones no lineales, lo que supuso un gran avance en el desarrollo de este campo. La importancia de este hecho fue tal que hoy en día sigue considerándose uno de los algoritmos más viables para recalcular los pesos en redes neuronales (ver Figura 37).

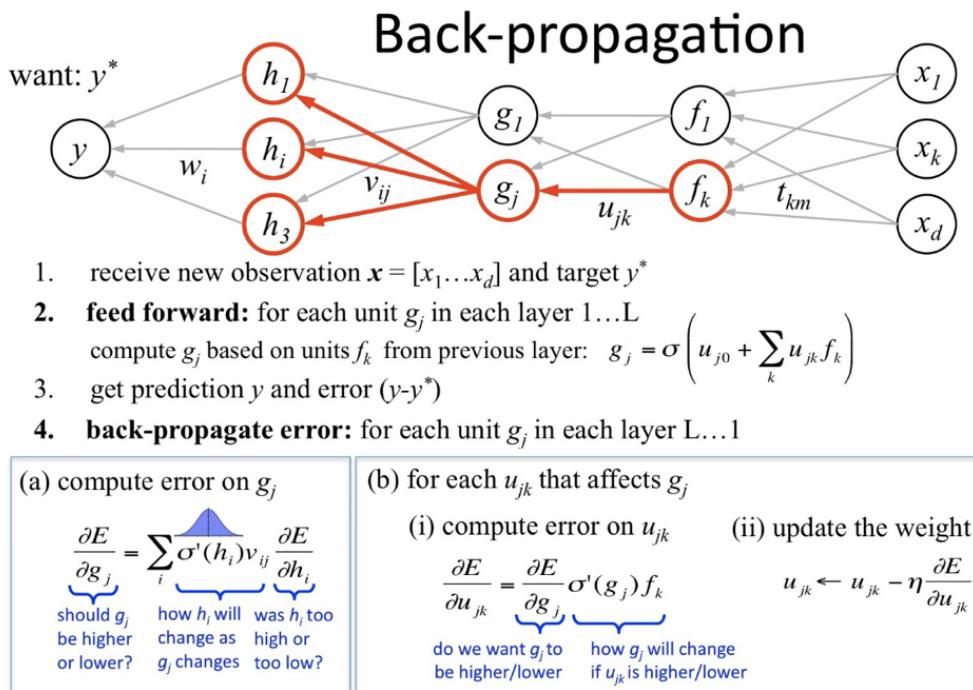


Figura 37: Representación del Perceptrón multicapa y el algoritmo Backpropagation²⁵

En 1989, Kurt, Max y Halbert White, demostraron matemáticamente que las redes neuronales multicapa son aproximaciones universales [8], lo que prueba que con el uso de múltiples capas ocultas una red neuronal es capaz de modelar cualquier función matemática, incluida la función XOR. Este hecho hizo que resurgiera su popularidad dentro de la comunidad científica.

Ese mismo año, Yann LeCun y sus colaboradores presentaron una de las primeras aplicaciones de redes neuronales en el mundo real [9]. Creó una red denominada LeNet capaz de clasificar imágenes de dígitos escritos a mano con una tasa de error aproximada del 5 %.

La estructura de LeNet incluía una capa convolucional como primera capa oculta, como se muestra de forma más clara en la Figura 38. En lugar de asignar un peso a cada píxel, esta capa utilizaba un pequeño conjunto de pesos que formaban un filtro de convolución para extraer características de la imagen de entrada, como la detección de vértices o líneas. La siguiente capa

²⁵ https://miro.medium.com/v2/resize:fit:1400/1*ZpFxEmEkigpz3AzrmZYAQ.png

se denomina pooling y reduce las características extraídas agrupando los píxeles por vecindad y comprimiéndolos en un solo valor con el objetivo de mantener solo la información más relevante.

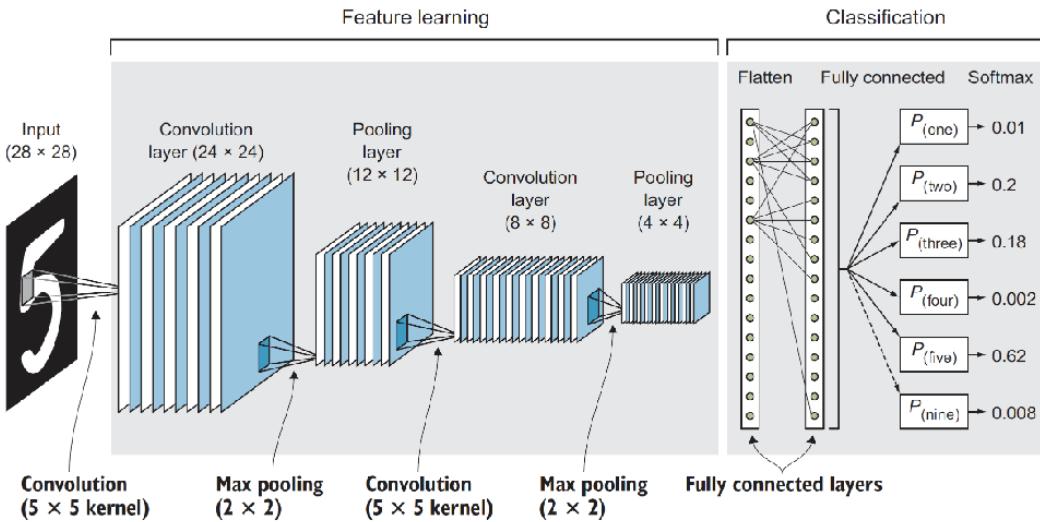


Figura 38: Arquitectura LeNet²⁶

Después de concatenar dos pares de estas capas, la red se conectaba a otra red multicapa más convencional en la que se clasificaban los dígitos basándose en las características extraídas, obteniendo en la salida una serie de nodos que también clasificaban los dígitos.

Durante los años 90, las técnicas de Machine Learning avanzaron y las redes neuronales comenzaron a utilizarse en nuevas áreas. Se adoptó un enfoque orientado a programas de análisis de datos para extraer conclusiones de ellos con buenos resultados. El éxito que estaban presentando impulsó propuestas cada vez más ambiciosas, como la de aplicarlas al procesamiento del habla y la toma de decisiones. Sin embargo, en 1991, Sepp Hochreiter demostró que entrenar redes neuronales muy profundas era extremadamente complejo [10]. El problema radicaba en que el error que llegaba a las capas superficiales era tan insignificante que no permitía ajustar los pesos de forma efectiva.

La aparición de técnicas alternativas como los árboles de decisión y las SVM, que lograban buenos resultados sin el elevado coste computacional del entrenamiento de las redes neuronales, provocó que el machine learning volviera a caer en el desinterés durante varios años.

A.3.3 Etapa Deep Learning (2006 – Actualidad)

No fue hasta 2006 y 2007 que el machine learning volvió a resurgir. Geoffrey Hinton introdujo las Deep Belief Networks (DBN) [11] y utilizó por primera vez el término Deep Learning.

La propuesta de Hinton era un método innovador para entrenar de manera efectiva redes neuronales profundas mediante una estrategia llamada greedy layer-wise pretraining. Consistía

²⁶ https://miro.medium.com/v2/resize:fit:1400/1*bGjusyTjh2SACnkkMHU_hA.png

en preentrenar cada capa de la red mediante un aprendizaje no supervisado utilizando Restricted Boltzmann Machines (RBM) [12], variantes de Boltzmann Machines con restricciones en las conexiones formando un grafo bipartito. Una vez preentrenado el modelo, se aplicaban los algoritmos de backpropagation para ajustar los pesos de la red de manera más eficiente.

El interés en las redes neuronales profundas, que había resurgido con el trabajo de Geoffrey, realmente despegó en 2012. Ese año, unos estudiantes de doctorado bajo la supervisión de Hinton presentaron AlexNet en la competición ImageNet [13] (ver Figura 39). El objetivo de la competición era etiquetar imágenes en mil categorías diferentes a partir de un conjunto de datos con millones de imágenes. Lo impresionante de este modelo fue que logró reducir el error en la clasificación de imágenes significativamente, bajándolo del 26% a alrededor del 16%. Este hito llamó la atención de muchos investigadores y grandes empresas.

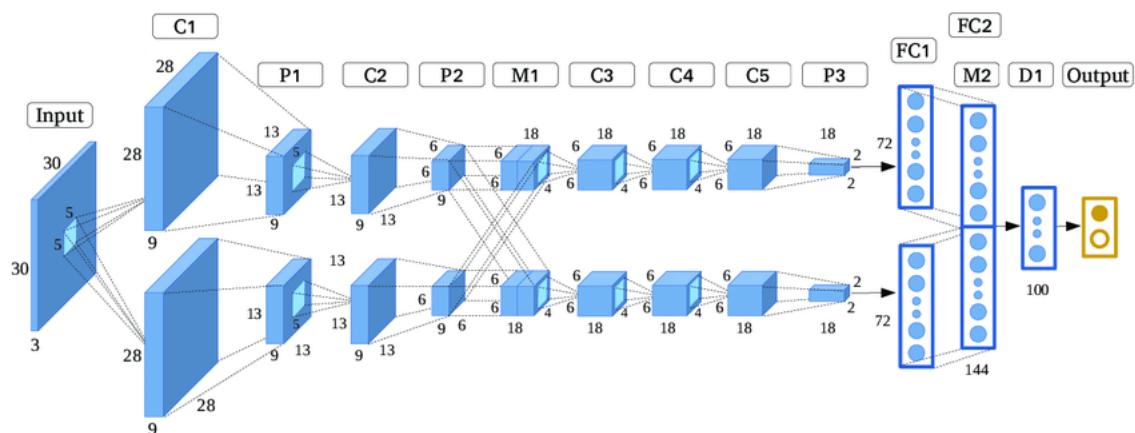


Figura 39: Arquitectura AlexNet²⁷

El interés no hizo más que crecer y, en 2014, Google presentó GoogLeNet, que incluía un módulo llamado Inception [14]. La novedad que este módulo aportaba era la posibilidad de realizar convoluciones de diferentes tamaños en paralelo, lo que ayudaba a encontrar el tamaño de kernel ideal. Así se consiguió reducir el error medio al 6,7% en la competición de ImageNet de ese mismo año.

Luego, en 2015, Microsoft creó ResNet [15], una nueva red que introducía una modificación: conectar la capa de salida a otra capa no inmediata para permitir transmitir el error a capas más superficiales. Esto hizo que, en la ImageNet de ese año, el error medio se redujera al 3,6 %, siendo la primera vez que se obtenían valores inferiores al error humano, que se considera aproximado al 5 %.

²⁷ [https://www.researchgate.net/publication/320723863/figure/fig4/\[...\]](https://www.researchgate.net/publication/320723863/figure/fig4/[...])

Anexo teórico B Redes neuronales

Las redes neuronales artificiales han emergido como una de las técnicas más revolucionarias dentro del campo del aprendizaje profundo. Inspiradas en la estructura y el funcionamiento del cerebro humano, estas redes están compuestas por unidades de procesamiento llamadas *neuronas artificiales*, organizadas en capas interconectadas que permiten modelar relaciones complejas en los datos. Su capacidad de aprendizaje ha sido fundamental en múltiples aplicaciones, desde el reconocimiento de imágenes hasta el procesamiento del lenguaje natural, impulsando avances significativos en inteligencia artificial. En este anexo se presentan los fundamentos de las redes neuronales, abordando su evolución, estructura y funcionamiento.

B.1 Fundamentos

El desarrollo de las redes neuronales artificiales se basa en principios matemáticos y computacionales que permiten crear modelos capaces de aprender patrones a partir de los datos. Para comprender su funcionamiento, es fundamental conocer los conceptos básicos en los que se basan, que expondremos a continuación.

B.1.1 Perceptrón

Para conocer los orígenes del perceptrón, es preciso retrotraerse al año 1943, un período que precede significativamente la conceptualización del término inteligencia artificial. Fue en dicho año cuando Warren McCulloch y Walter Pitts divulgaron un modelo matemático que procuraba representar de manera simplificada el funcionamiento de una neurona biológica [2]. Este modelo se fundamenta en una estructura lógica que procesa la información mediante entradas y salidas binarias, sentando así los cimientos para el desarrollo de las redes neuronales artificiales.

La neurona biológica, en su configuración más básica, se constituye a partir de tres componentes principales: el soma o cuerpo celular, que alberga el núcleo y regula la actividad neuronal derivada de los impulsos nerviosos recibidos de otras neuronas a través de los canales de entrada, denominadas dentrinas. Estas extensiones ramificadas, juegan un papel crucial en la transmisión de los impulsos nerviosos. Cuando el impulso recibido supera un umbral determinado, la neurona se activa, generando un impulso a través del canal de salida, el axón. Éste último se conecta a las dentrinas de otra neurona mediante lo que se conoce como conexión sináptica, permitiendo la transferencia de información entre las células neuronales. La comunicación entre neuronas se lleva a cabo mediante señales eléctricas dentro de la célula y señales químicas en las sinapsis, lo que permite el procesamiento de información en el cerebro y el sistema nervioso (ver Figura 40).

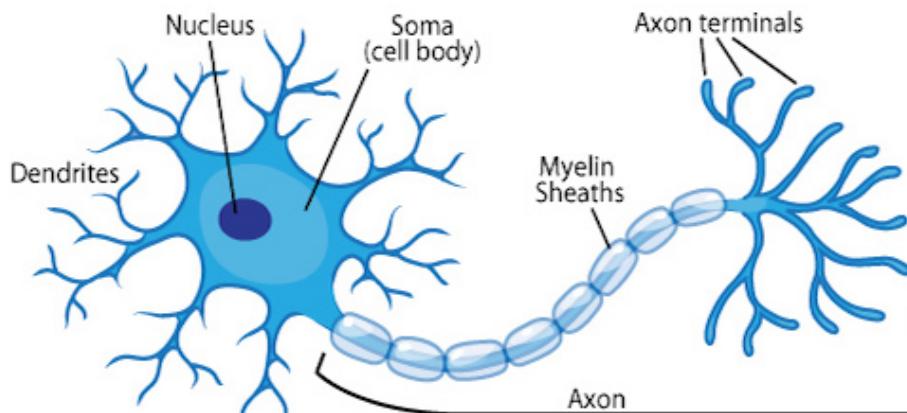


Figura 40: Anatomía de una neurona biológica²⁸

En este sentido, McCulloch y Pitts desarrollaron su modelo matemático, el cual recibe dos tipos de entradas binarias: excitadoras (x_i) e inhibidoras (x_j^*). Dichas entradas son procesadas mediante la suma de estas. Si dicha suma supera un umbral prefijado, la neurona se activará y su salida valdrá uno. En caso de no superar el umbral, su salida valdrá cero. Es importante destacar que, en presencia de alguna entrada inhibidora activa, la salida tomará el valor cero, independientemente de si la agregación excede el umbral.

Este primer modelo se caracteriza por su limitación a las funciones booleanas, determinada por su naturaleza intrínseca. Adicionalmente, carece de un mecanismo para el autoajuste del valor del umbral, el cual debe ser definido de antemano. Asimismo, es posible que no sea deseable que todas las entradas sean iguales, sino que se prefiera otorgar mayor importancia a algunas entradas frente a otras.

En 1958, Frank Rosenblatt desarrolló el perceptrón con el propósito de resolver las limitaciones observadas en la neurona de McCulloch-Pitts [4]. Este modelo posee la capacidad de procesar cualquier entrada real, lo que resulta en la eliminación de las entradas inhibidoras. En consecuencia, una entrada negativa generaría un comportamiento equivalente. Este modelo opera recibiendo múltiples entradas, cada una con un peso asociado, que se suman y procesan mediante una función de activación, la cual, a diferencia del anterior modelo, ya no se encuentra en la propia neurona sino como elemento posterior. En contraste con el modelo de McCulloch-Pitts, el perceptrón puede modificar sus pesos a través de un algoritmo de aprendizaje supervisado (ver Figura 41).

²⁸ <https://ml4a.github.io/images/neuron-anatomy.jpg>

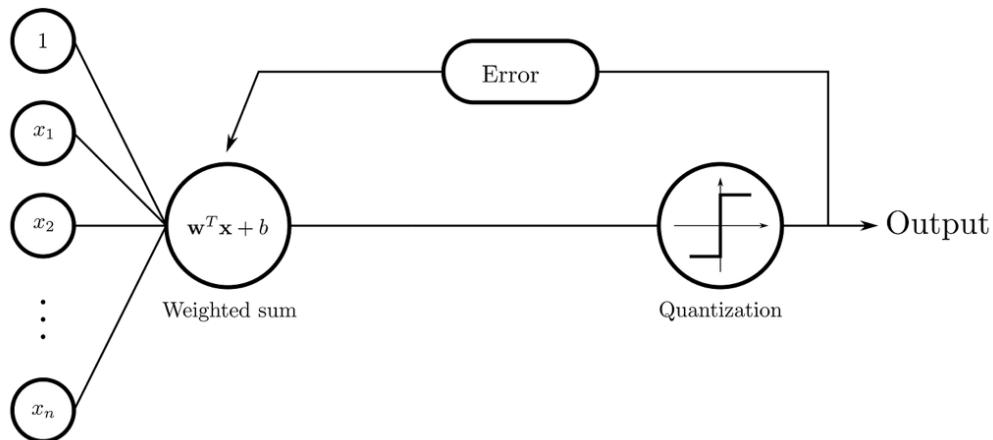


Figura 41: Perceptrón y su proceso de aprendizaje²⁹

En 1960, Bernard Widrow y Marcian Hoff introdujeron el modelo ADALINE (Adaptive Linear Neuron) [5], que supuso una mejora significativa sobre el perceptrón al emplear una función de activación lineal en lugar de una función escalón. Este cambio permitió la aplicación de técnicas de optimización para ajustar los pesos de manera más eficiente. A diferencia del perceptrón, que ajusta los pesos únicamente en función del resultado final de la clasificación, el modelo ADALINE los ajusta considerando la diferencia entre la salida deseada y la salida real antes de la aplicación de la función de activación. Estas mejoras contribuyeron significativamente al desarrollo de la versión actual del perceptrón.

B.1.2 Funciones de activación

Como se ha expuesto en la sección precedente, el desarrollo del perceptrón implica que la función de activación deja de ser parte integrante del mismo, pasando a ser un elemento posterior que actúa sobre su salida. Este cambio permite la modificación de la salida del perceptrón para que esta tenga sentido en el contexto del problema a resolver.

Como se ha mencionado anteriormente, se ha expuesto el modo en el que las primeras aplicaciones utilizaban la función booleana. Posteriormente, la neurona de McCulloch-Pitts hacía uso de la función escalonada. Este paradigma cambió con la llegada del perceptrón, que trajo consigo el uso de valores reales. Aunque teóricamente podría utilizarse cualquier función, se expondrán las más populares, en la mostramos las más populares.

²⁹ https://miro.medium.com/v2/resize:fit:1400/1*BM8HuBXy9XYLvSGlBZwL0g.png

Tabla 14: Principales funciones de activación

Nombre	Gráfica	Ecuación
Escalonada		$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$
Sigmoide		$f(x) = \frac{1}{1 + e^{-x}}$
Identidad		$f(x) = x$
ReLU		$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$
PReLU		$f(x) = \begin{cases} ax, & x < 0 \\ x, & x \geq 0 \end{cases}$
TanH		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

B.1.3 Tipos de capas

Para proceder a la descripción de la organización y las conexiones de las neuronas que constituyen una red neuronal, es imperativo iniciar la investigación con el primer componente, que es la capa. Se denomina capa a la agrupación de neuronas de una red que reciben las mismas entradas, como se muestra en la Figura 42.

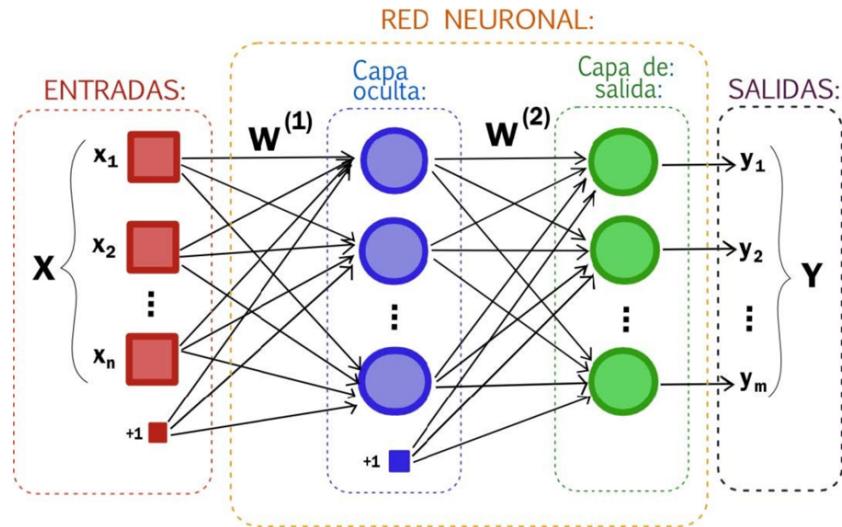


Figura 42: Estructura básica de una red neuronal multicapa³⁰

Como se ha expuesto en investigaciones previas, se ha determinado que la ecuación de un perceptrón en formato vectorial es:

$$y = W^T x \quad (2)$$

En términos generales, las n entradas se organizan en un vector $x = [x_0, x_1, \dots, x_n]^T$, donde el primer componen $x_0 = 1$. Por otro lado, las salidas de las m neuronas de la capa se agrupan en el vector salida de la capa $y = [y_0, y_1, \dots, y_m]^T$. En lo que respecta a los pesos de cada neurona, $w^i = [w_0^i, w_1^i, \dots, w_n^i]^T$, son agrupados en la matriz de pesos $W \in \mathbb{R}^{(n+1) \times m}$, donde $(n + 1)$ representa el número de entradas más el sesgo. Como podemos observar en la **¡Error! No se encuentra el origen de la referencia.**, se ha omitido la inclusión de la función de activación, dado que por lo general las neuronas de una misma capa cuentan con las mismas conexiones y función de activación y que ésta se aplica al valor de cada salida por separado, podemos reescribir la ecuación de forma que la función se aplica a cada componente del vector:

$$y = f(W^T x) \quad (3)$$

Una vez que se han introducido las ecuaciones y sus componentes, se puede proceder a la introducción de los tres tipos de capas.

³⁰ [https://media.linkedin.com/dms/image/D4E12AQHGrL3AHLPo2Q/\[...\]](https://media.linkedin.com/dms/image/D4E12AQHGrL3AHLPo2Q/[...])

- **Capa de entrada:** En el ámbito de la inteligencia artificial, es una práctica común representar los datos de entrada como una capa adicional de la red neuronal, lo que se conoce como *capa 0*, donde los elementos de la capa son los componentes del vector $x = [x_0, x_1, \dots, x_n]^T$. Por otro lado, la salida de esta capa y^0 , se corresponde directamente con el vector de entrada x , haciendo que la utilización de esta capa simplifique la notación de las operaciones.
- **Capa oculta y Capa de salida:** Capas que se ubican posteriores a la capa de entrada. Las neuronas que conforman estas capas han sido expuestas en el apartado **¡Error! No se encuentra el origen de la referencia.**, mediante el estudio de los perceptrones. Estos componentes poseen pesos que desempeñan un papel crucial en la transformación de los datos de entrada, junto con la función de activación que determina su comportamiento y respuesta. La principal diferencia entre la capa oculta y la capa de salida radica en la ubicación dentro de la red neuronal. Específicamente, la capa de salida constituye la última capa de la red, mientras que la capa oculta se ubica entre la capa de entrada y la capa de salida.

B.1.4 Aplicación a problemas de clasificación

Las redes neuronales exhiben la ventaja de poseer múltiples neuronas en la capa de salida, lo que facilita la resolución de problemas que resultan inviables de ser abordados por un solo perceptrón, como es el caso de la clasificación en múltiples categorías.

A modo ilustrativo, en el contexto del reconocimiento de dígitos en imágenes, se podría optar por la implementación de diez neuronas en la capa de salida, donde cada una estaría encargada de identificar un número específico dentro de la imagen ingresada. Sin embargo, no se puede asegurar que solo una neurona se active o que ninguna lo haga.

Para gestionar esta situación, se implementa la función de activación identidad en la capa de salida y se aplica la función softmax al vector de salida. Ésta última es una generalización de la función sigmoide, que transforma el vector de salida en una distribución de probabilidad.

$$S(y_i) = \frac{e^{y_i}}{\sum_{j=1}^n e^{y_j}} \quad (4)$$

El empleo de esta función permite interpretar la salida de cada neurona como una probabilidad. No obstante, esto no implica que la red haya llevado a cabo una clasificación definitiva para la entrada. Para obtener una categoría concreta, es preciso convertir el vector de probabilidades en una representación que refleje la clase asignada. Por lo general, esto se realiza seleccionando la clase correspondiente a la neurona con la mayor probabilidad. Un método común para lograrlo es la codificación one-hot, que consiste en asignar el valor 1 a la componente del vector con la probabilidad más alta (en caso de empate, se elige una de ellas) y 0 a las demás.

$$y = \begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix} \xrightarrow{\text{one-hot}} \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

Figura 43: Transformación softmax y codificación one-hot

Al examinar la Figura 43, se evidencia que el vector de salida, al ser sometido a la aplicación de la transformación softmax, da lugar a una representación de distribución de probabilidad. En consecuencia, al codificar este vector de probabilidades mediante el método de one-hot, se obtiene la clasificación de la entrada, entendida como la clase correspondiente al valor 1 asignado a dicho vector.

B.1.5 Aplicación a imágenes

En el campo de la inteligencia artificial, las redes neuronales han demostrado su capacidad para abordar problemas de gran complejidad, que trascienden la mera clasificación multiclase. Al representar una imagen como un vector de píxeles, esta puede ser procesada por una red neuronal, pudiendo incluso generar una imagen como salida. La clave del proceso radica en la utilización de una cantidad adecuada de neuronas en la capa de salida, proporcional al número de píxeles que se desean obtener.

Es crucial reconocer que una de las restricciones fundamentales de las redes neuronales radica en la necesidad de que tanto la entrada como la salida estén representadas como vectores. En consecuencia, para procesar datos con estructuras espaciales, como las imágenes, es imperativo transformarlos en una representación vectorial. Este procedimiento se denomina flattening o aplastamiento.

Este proceso implica la transformación de una estructura de datos multidimensional, como una imagen, en un vector unidimensional para su procesamiento por parte de una red neuronal. En lo que respecta a su aplicación en imágenes, que comúnmente se representan como matrices de píxeles con una o más dimensiones (como en imágenes en escala de grises o en color con canales RGB), el proceso de flattening reorganiza estos datos en una única fila de valores (ver Figura 44) . Este paso resulta de vital importancia en el contexto de las redes neuronales con capas fully connected, ya que estas requieren que la información de entrada esté en formato vectorial. No obstante, en arquitecturas especializadas, tales como las CNNs, se busca evitar el flattening en etapas intermedias con el fin de preservar la estructura espacial de los datos y mejorar el rendimiento en tareas como el reconocimiento de imágenes.

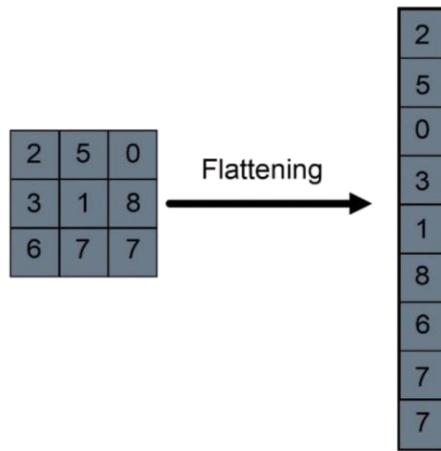


Figura 44: Ejemplo de flattening³¹

B.2 Desarrollo de redes neuronales

Una vez examinados los componentes, aplicaciones y limitaciones de las redes neuronales, el siguiente paso a considerar es el desarrollo de estas. Para construir una red neuronal, es fundamental definir primero su arquitectura, lo cual generalmente se realiza mediante un proceso de prueba y error. La estrategia más común es basarse en arquitecturas populares que han demostrado buenos resultados en diversos problemas. Otro aspecto esencial en la creación y aplicación de redes neuronales es el manejo de los datos. El análisis exhaustivo de los problemas abordados con estas redes revela su esencia como un proceso de aproximación de funciones, donde la función objetivo se modela a partir de un conjunto de ejemplos representativos.

En el ámbito de la investigación en Machine Learning, se ha observado una tendencia a distinguir entre la fase de obtención del modelo y la fase de uso de este. Este enfoque metodológico se ha convertido en una práctica común en el desarrollo de modelos de Machine Learning, con el objetivo de facilitar la comprensión y la aplicación de estos sistemas en diversos contextos. Como se refleja en la Figura 45.

³¹ [https://www.researchgate.net/publication/359174861/figure/fig5/\[...\]](https://www.researchgate.net/publication/359174861/figure/fig5/[...])

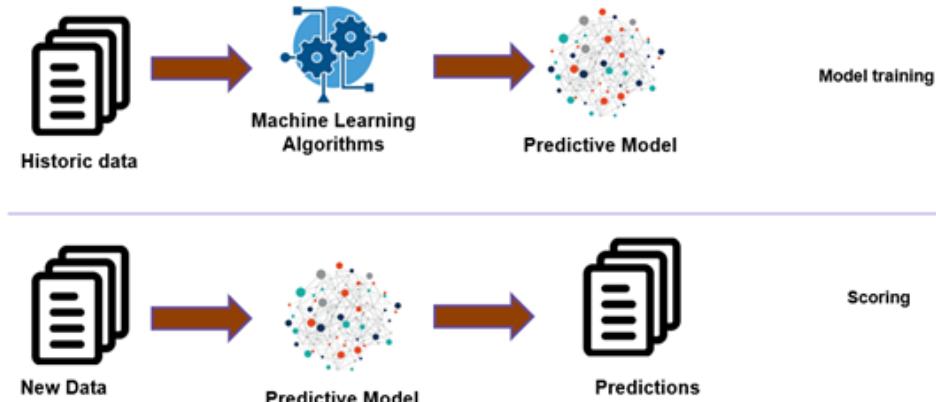


Figura 45: Etapas de entrenamiento e inferencia de un modelo de Machine Learning³²

- **Etapa de entrenamiento:** La etapa de entrenamiento constituye un componente esencial en el desarrollo de una red neuronal, dado que en esta fase la red aún no ha establecido sus pesos. Para ajustarlos, se emplean los datos disponibles con el propósito de que la red alcance la aproximación más precisa posible a la función deseada. Este ajuste de pesos puede llevarse a cabo de diversas maneras: manualmente, de manera aleatoria o de forma automática mediante un algoritmo de entrenamiento. En el contexto de las redes neuronales, esta fase se denomina etapa de aprendizaje, durante la cual la red optimiza sus parámetros para mejorar su rendimiento en la tarea específica. El resultado de este proceso es una red neuronal con un conjunto de pesos definidos que le permiten realizar predicciones o clasificaciones con mayor precisión.
- **Etapa de inferencia:** La etapa de inferencia en una red neuronal se produce cuando los pesos dejan de ser actualizados y la red es utilizada exclusivamente para realizar predicciones. Esta fase resulta esencial para evaluar la efectividad del entrenamiento y determinar si los resultados obtenidos son satisfactorios o si es necesario continuar ajustando la red. Una de las principales ventajas de las redes neuronales radica en su capacidad para generar salidas a partir de cualquier dato de entrada, no únicamente aquellos con los que fueron entrenadas. Esta capacidad permite que la red procese información nueva y realice predicciones sobre datos nunca vistos previamente. En esta etapa, la red no se ve sometida a un proceso de aprendizaje o modificación de sus parámetros, sino que simplemente aplica el conocimiento adquirido durante el entrenamiento para inferir resultados. El único producto de esta fase es la salida generada por la red a partir de los datos de entrada proporcionados.

Para asegurar el correcto funcionamiento de una red neuronal, resulta imperativo disponer de procedimientos que posibiliten la evaluación de su rendimiento. Las métricas de evaluación satisfacen esta necesidad, puesto que no solo facilitan el monitoreo del entrenamiento, sino que también brindan una medida objetiva para la comparación de redes con diferentes pesos o arquitecturas.

³² <https://blogger.googleusercontent.com/img/b/...>

Para calcular estas métricas, es imperativo conocer la clase o el valor real de cada dato de entrada, ya que se requiere comparar la salida esperada con la salida generada por la red. En consecuencia, las métricas de evaluación no pueden aplicarse durante la fase de inferencia, ya que en esta etapa la red procesa datos sin una referencia conocida para validar sus respuestas. En contraste, estas métricas pueden ser utilizadas en los conjuntos de datos de entrenamiento, validación y prueba, donde se cuenta con etiquetas o valores de referencia que permiten medir con precisión el rendimiento de la red.

Como se ha mencionado anteriormente, para entrenar y evaluar de manera efectiva una red neuronal, es común dividir los datos en tres conjuntos principales, cada uno con un propósito específico:

- **Conjunto de entrenamiento:** este conjunto de datos se utiliza para ajustar los pesos de las neuronas durante la etapa de entrenamiento. La red aprende a reconocer patrones a partir de estos ejemplos, optimizando su desempeño en la tarea asignada.
- **Conjunto de validación:** evalúa el comportamiento de la red utilizando datos que no han sido observados anteriormente. A diferencia del conjunto de entrenamiento, estos datos no se utilizan para ajustar los pesos de la red, sino para medir su rendimiento durante el proceso de entrenamiento y para tomar decisiones sobre ajustes en la arquitectura o hiperparámetros.
- **Conjunto de prueba:** se utiliza para evaluar el rendimiento final del modelo una vez completado el entrenamiento. A diferencia del conjunto de validación, este conjunto no se emplea en ninguna fase del ajuste del modelo, sino que se usa únicamente al final para estimar el comportamiento que tendrá la red durante la fase de inferencia.

Durante el proceso de entrenamiento, la red es sometida a múltiples evaluaciones utilizando el conjunto de validación, conforme a los parámetros establecidos, con el propósito de monitorear su rendimiento. En caso de que se evidencie una estabilización o incluso un deterioro en las métricas de rendimiento, se recomienda interrumpir el entrenamiento y considerar la implementación de ajustes en la estrategia o la revisión del código en busca de errores. Además, la evaluación en el conjunto de entrenamiento sirve como referencia para compararla con la de validación, ya que se espera que ambas métricas sean similares. Si la diferencia entre ellas es significativa, puede indicar problemas como sobreajuste o subajuste.

B.2.1 Métricas de evaluación

Las métricas de evaluación no son universales y exhiben variaciones en función del tipo de problema que se esté abordando. En el contexto de los problemas de clasificación, las métricas predominantes se fundamentan en la interpretación del vector de salida de la red, donde la asignación de la clase se determina predominantemente según la posición del valor máximo en dicho vector. No obstante, este enfoque presenta una limitación fundamental:

existe la posibilidad de que una entrada sea clasificada incorrectamente en una categoría, incluso si la probabilidad asignada es demasiado baja.

Para abordar esta problemática, se implementa un umbral, como 0.5, que delimita el rango de probabilidad. En caso de que el valor de probabilidad más alto en la salida supere este umbral, se asigna la clase correspondiente. Sin embargo, es importante destacar que este procedimiento no garantiza una clasificación exacta, ya que el valor máximo puede situarse por debajo del umbral o, incluso, no coincidir con la clase esperada, en caso de que supere el umbral. En consecuencia, en múltiples circunstancias se implementan métricas complementarias, tales como la precisión, la exhaustividad o la métrica F1, con el propósito de obtener una valoración más precisa del rendimiento del modelo.

La evaluación del rendimiento de una red neuronal se fundamenta en la comparación entre las clases predichas por la red y las clases esperadas. Mediante esta comparación, es posible obtener cuatro métricas fundamentales:

- **True Positives (TP):** Se produce cuando la red clasifica correctamente una instancia como positiva, es decir, cuando la clase asignada por la red coincide con la clase positiva esperada.
- **True Negatives (TN):** Se produce cuando la red identifica correctamente una instancia como negativa, es decir, la clase asignada como negativa coincide con la clase negativa esperada.
- **False Positives (FP):** Suceden cuando la red clasifica erróneamente una instancia como positiva cuando en realidad pertenece a la clase negativa. Este error se conoce como falso positivo o error tipo I.
- **False Negatives (FN):** Se producen cuando la red clasifica incorrectamente una instancia como negativa cuando en realidad pertenece a la clase positiva. Este error se conoce como falso negativo o error de tipo II.

Estas métricas se organizan en una estructura denominada matriz de confusión, la cual funciona como un contador que permite visualizar el rendimiento de la red en la clasificación de los datos. En esta matriz, las filas representan las clases reales (esperadas) de las instancias de entrada, mientras que las columnas corresponden a las clases predichas por el modelo. En este contexto, cada predicción realizada por la red se traduce en un incremento del valor de la celda correspondiente, según la coincidencia o el error entre la clase esperada y la clase predicha. Es pertinente señalar que la construcción de la matriz de confusión puede abarcar todas las clases contenidas en el modelo (ver Figura 46).

	Walleye	Largemouth Bass	Bluegill	Rainbow Trout
Actual value	TP			
Walleye		TP		
Largemouth Bass			TP	
Bluegill				TP
Rainbow Trout				TP
Predicted value	Walleye	Largemouth Bass	Bluegill	Rainbow Trout

Figura 46: Ejemplo de matriz de confusión para 4 clases³³

En problemas de clasificación multiclas, si la salida de la red se encuentra en formato one-hot encoding o si se emplea un umbral de probabilidad, es posible generar una matriz de confusión para cada clase. En este caso, se considera la clase de interés como la clase positiva y el resto de las clases como negativas. Este enfoque, denominado método de matrices de confusión individuales por clase, es el más prevalente y posibilita la evaluación del rendimiento del modelo en cada categoría de manera independiente (ver Figura 47).

	Positive	Negative
Actual value	TP	FN
Negative	FP	TN
Predicted value	Positive	Negative

Figura 47: Matriz de confusión considerando únicamente las clases positivas y negativas³⁴

El análisis de la matriz de confusión proporciona información relevante sobre los errores de clasificación, lo que permite optimizar el modelo para mejorar su precisión y capacidad de generalización.

Una vez generada la matriz de confusión para cada categoría, es posible estimar diversas métricas de evaluación que permiten la valoración del rendimiento del modelo de clasificación. Las métricas más frecuentemente empleadas incluyen:

³³ <https://www.ibm.com/content/dam/connectedassets-adobe-cms/...>

³⁴ <https://www.ibm.com/content/dam/connectedassets-adobe-cms/...>

- **Precision:** constituye una métrica que permite evaluar la proporción de predicciones correctas entre todas las predicciones positivas emitidas por el modelo. Este cálculo se determina mediante la siguiente fórmula:

$$Precision = \frac{TP}{TP + FP} \quad (5)$$

Un alto nivel de precisión indica que el modelo presenta un número reducido de falsos positivos.

- **Recall:** También conocido como *Sensitivity*, se refiere a la aptitud del modelo para identificar de manera precisa las instancias positivas. Su cálculo se determina mediante la siguiente fórmula:

$$Recall = \frac{TP}{TP + FN} \quad (6)$$

Un alto índice de recall indica que el modelo detecta la mayoría de los casos positivos, aunque puede resultar en una mayor cantidad de falsos positivos.

- **Specificity:** evalúa la capacidad del modelo para identificar correctamente las instancias negativas. Se define como:

$$Specificity = \frac{TN}{TN + FP} \quad (7)$$

Un modelo con alta especificidad tiene pocos falsos positivos.

- **F1 Score:** La métrica en cuestión constituye la media armónica entre precisión y recall, proporcionando un equilibrio entre ambas métricas. Su cálculo se determina mediante la siguiente fórmula:

$$F1\ Score = \frac{Precision \times Recall}{Precision + Recall} \quad (8)$$

La relevancia de esta métrica radica en su capacidad para alcanzar un balance óptimo entre precisión y recall, lo que la convierte en un instrumento de gran utilidad en diversos contextos.

- **Accuracy:** expresa el porcentaje de predicciones correctas sobre el total de predicciones efectuadas. Su cálculo se determina mediante la siguiente fórmula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

Si bien constituye una métrica global, puede resultar menos representativa en conjuntos de datos desbalanceados.

El inconveniente principal asociado a las métricas de clasificación radica en su dependencia de la selección de un valor umbral para la determinación de la pertenencia de una instancia a la clase positiva o negativa. La elección de dicho umbral constituye un aspecto de suma importancia, puesto que incide de manera directa en el equilibrio entre las distintas métricas de evaluación.

En el caso de que se determine un umbral bajo, el modelo en cuestión clasificará un mayor número de instancias como positivas, incrementando así la sensibilidad. Este fenómeno implica que el modelo identificará la mayoría de los casos positivos, aunque también puede generar un alto número de falsos positivos.

Por el contrario, al establecer un umbral más elevado, se reducirá la cantidad de falsos positivos, incrementando la especificidad y asegurando que solo los casos con alta probabilidad sean clasificados como positivos. No obstante, esta estrategia puede conducir a un incremento en los falsos negativos, lo que implica que ciertos casos positivos podrían pasar desapercibidos para el modelo.

La sensibilidad y la especificidad se erigen como métricas opuestas que definen el comportamiento del modelo y su idoneidad para un problema específico. En contextos donde prevalece la necesidad de minimizar los falsos negativos, como en el diagnóstico de enfermedades, se tiende a optar por un umbral bajo que garantice una alta sensibilidad. En contraste, en contextos donde se prioriza la minimización de los falsos positivos, tales como en la detección de fraudes, se recomienda establecer un umbral alto para maximizar la especificidad.

Cuando se procede a la comparación de distintos modelos sin la necesidad de establecer un umbral concreto, se hace factible la utilización de métricas como el Área Bajo la Curva ROC (AUC-ROC). Esta métrica permite la evaluación del rendimiento global del modelo, mediante el análisis de la relación entre la sensibilidad y la especificidad, trazando la curva ROC (Receiver Operating Characteristic).

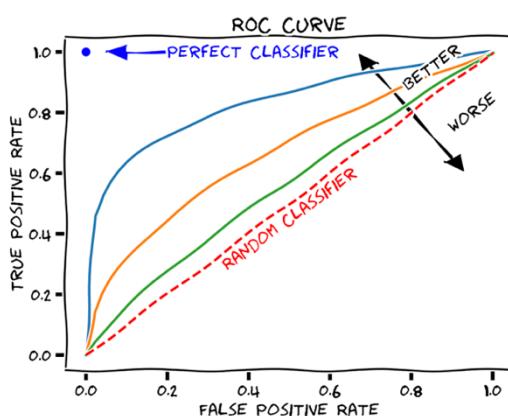


Figura 48: Ejemplo de gráfica de Curva ROC³⁵

³⁵ <https://polmartisanahuja.com/wp-content/uploads/2021/01/word-image.png>

Esta curva representa la tasa de verdaderos positivos (sensibilidad) en el eje Y, y la tasa de falsos positivos (especificidad) en el eje X, como puede verse en la Figura 48. La ubicación de la curva en el espacio de decisión es relevante para la evaluación de la calidad del modelo. Un posicionamiento elevado y hacia la izquierda indica un modelo de mayor eficacia, ya que implica una detección precisa de los positivos y una minimización de los falsos positivos.

B.2.2 Estimación del error

El propósito esencial de la etapa de entrenamiento en una red neuronal radica en la identificación de los pesos óptimos que faciliten la realización de predicciones precisas por parte de la red, empleando el conjunto de datos de entrenamiento. La modificación de dichos pesos requiere la evaluación de la discrepancia entre las predicciones de la red y los valores esperados, lo que se denomina error de la red.

No obstante, no todos los errores tienen el mismo impacto en el rendimiento del modelo. En consecuencia, se implementa una estimación que mide la gravedad de ciertos valores de error, denominada pérdida. La función matemática encargada de calcular dicha pérdida, a partir de la diferencia entre la salida de la red y la salida esperada, se denomina función de error, función de pérdida o función de coste.

Existen diversas funciones de pérdida y su eficacia dependerá del tipo de problema que estemos abordando, pero en general cualquier función derivable podría servir como loss function puesto que nuestro objetivo será minimizarla. Por ejemplo, para problemas de clasificación una de las funciones más comúnmente utilizadas es:

- **Cross-Entropy (CE):** también conocida como *log error*, es especialmente útil en clasificaciones donde la salida representa la probabilidad de pertenencia a una clase, con valores entre 0 y 1. En problemas de clasificación binaria, donde solo existen dos clases, esta función también se conoce como *Binary Crossentropy (BCE)* y se formula de la siguiente manera:

$$BCE = - \sum_{i=1}^n [y_i * \ln(p_i) + (1 - y_i) * \ln(1 - p_i)] \quad (10)$$

Siendo n el número de ejemplos, y_i la salida esperada y p_i la salida de la red, es decir, probabilidad de que el ejemplo pertenezca a la clase positiva.

- **Categorical Crossentropy (CCE):** Esta función utilizada en problemas de multiclase, calcula la probabilidad de pertenecer a cada una de las clases, se calcula de la siguiente forma:

$$CCE = - \sum_{i=1}^n \sum_{j=1}^m y_j^i * \ln(p_j^i) \quad (11)$$

Siendo n el número de ejemplos, y_j^i sería la salida esperada para cada clase j , p_j^i salida predicha por el modelo para la clase j y m el número de clases del problema.

- **Mean Square Error (MSE):** También conocido como *L2 loss*, consiste en la media de las diferencias entre la salida esperada y la salida obtenida al cuadrado, siendo su fórmula:

$$MSE = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^m (p_j^i - y_j^i)^2 \quad (12)$$

B.2.3 Inicialización de los pesos

La asignación de valores iniciales apropiados constituye un paso crucial en el proceso de entrenamiento. Esta asignación, ya sea mediante la generación autónoma del diseño de la red o la utilización de una red preexistente como punto de partida, se erige como un componente fundamental para el entrenamiento efectivo de las redes neuronales. En este sentido, resulta imperativo asignar valores a todos los filtros de las capas de convolución y a las conexiones de las capas densas de la red, así como a cualquier capa que posea parámetros configurables.

Una de las propuestas más recurrentes es la de asignar ceros u otro valor específico en todos los pesos. Sin embargo, esta estrategia puede tener implicaciones indeseadas debido a la simetría. La aplicación de esta práctica a todos los pesos de la red resulta en una modificación uniforme que impide el aprendizaje efectivo. Ante esta situación, se han desarrollado técnicas de inicialización alternativas que introducen variabilidad en la red, permitiendo así el desarrollo de representaciones útiles de los datos y la mejora de la capacidad de generalización durante el entrenamiento.

Una segunda propuesta para evitar la simetría en la inicialización de los parámetros es realizarlo de forma aleatoria, aunque con cierto control, ya que, si los valores iniciales son demasiado pequeños, los valores de entrada que se propagan a lo largo de la red también serán muy pequeños y, por tanto, los valores que lleguen a las capas finales de la red no serán relevantes para el entrenamiento. De manera análoga, en el caso de valores excesivamente pequeños, si los valores iniciales son demasiado pequeños, los valores de entrada que se propagan a lo largo de la red también serán muy pequeños y, en consecuencia, los valores que alcancen las capas finales de la red no serán relevantes para el entrenamiento. De manera similar, en el caso de valores excesivamente grandes, si los valores iniciales son demasiado grandes, en las capas finales los valores serán demasiado elevados y dificultarán el aprendizaje.

Por tanto, se torna imperativo establecer un rango adecuado de valores iniciales que evite la simetría y que permita una propagación efectiva de la información a lo largo de la red, garantizando así un entrenamiento eficiente y una mayor capacidad de generalización de la red.

Para inicializar los parámetros con valores iniciales óptimos, es posible recurrir a dos métodos populares que no son puramente aleatorios:

- **Inicialización de Xavier/Glorot:** Esta técnica, presentada en 2009 por Xavier Glorot y Yoshua Bengio. Es una solución al problema de la inicialización aleatoria. El método inicializa los pesos de la red con una distribución uniforme de forma que la varianza y la desviación típica de estos sea igual a 1, teniendo en cuenta el número de unidades de la capa, de forma que su fórmula es:

$$w_j^i \sim N \left(0, \sqrt{\frac{1}{n}} \right) \quad (13)$$

- **Inicialización de He:** Método creado en 2015 por los creadores de ResNet, que ganó la competición de ImageNet, consiguiendo bajar el error poder debajo del valor del error humano (considerado como un 5%). Tienen en cuenta funciones de activación no lineales, en la que destaca la función ReLu. Este procedimiento posibilita la convergencia de modelos sumamente profundos, para los cuales la inicialización Xavier no resultaba efectiva. Su metodología se fundamenta en la asignación de un valor aleatorio tomado de una distribución uniforme:

$$w_j^i \sim U \left(0, \sqrt{\frac{2}{n}} \right) \quad (14)$$

Existe una alternativa a los métodos previamente mencionados, que consiste en la utilización de una red previamente entrenada en lugar de iniciar desde cero. Este procedimiento se basa en la premisa de que una red ha sido utilizada con éxito para resolver un problema complejo y general, o que un investigador ha compartido una red con pesos ya ajustados.

En lugar de iniciar el entrenamiento con una inicialización aleatoria para un nuevo problema, se aprovecha una red entrenada previamente, lo que permite reutilizar el conocimiento adquirido durante su entrenamiento y acelerar el aprendizaje. Este enfoque, denominado *transfer learning*, consiste en aplicar lo aprendido en un problema a otro, que suele ser más simple que el original. Este método permite que el entrenamiento comience desde una configuración más cercana al óptimo en comparación con una inicialización aleatoria, lo que resulta en una mejora en la eficiencia y el rendimiento del modelo.

B.2.4 Algoritmos de optimización

Como se explicó en apartados precedentes, el cálculo del error cometido permite el ajuste de los parámetros de la red. De este modo, el proceso se simplifica a un problema de optimización, en el que se deben encontrar los valores óptimos para cada parámetro de la red.

Para lograrlo, se procederá a la estimación de “la pendiente” de la función en un punto específico, o lo que es lo mismo, se efectuará el cálculo de la derivada de la función de error. Dicha pendiente muestra la dirección de máximo decremento del gradiente para cada uno de los pesos de la red, los cuales serán ajustados desde la última capa hacia las primeras con la intención de reducir el error en futuros ejemplos.

Como se ha mencionado anteriormente, la pendiente representa la dirección de la máxima disminución, la cual es calculada mediante la derivada parcial de cada uno de los coeficientes de la red, expresada en forma de vector:

$$\vec{\nabla}_{\theta} J(\theta) = \left(\frac{\partial J(\theta)}{\partial \theta_0}, \frac{\partial J(\theta)}{\partial \theta_1}, \dots, \frac{\partial J(\theta)}{\partial \theta_n} \right) \quad (15)$$

Siendo J la función de error, θ el conjunto de pesos que constituyen la red y n el número total de pesos de la red.

A partir de esto, se procede a la actualización de cada uno de los pesos de la red, mediante la adición o la sustracción, según lo indicado como el opuesto a la dirección del gradiente, máximo descenso, como se observa en la Figura 49. Este procedimiento da como resultado:

$$\theta_{nuevo} = \theta_{antiguo} - \alpha \nabla_{\theta} J(\theta) \quad (16)$$

Donde α es el *learning rate*, establece si el valor de la actualización del peso del parámetro se hace en mayor o en menor medida.

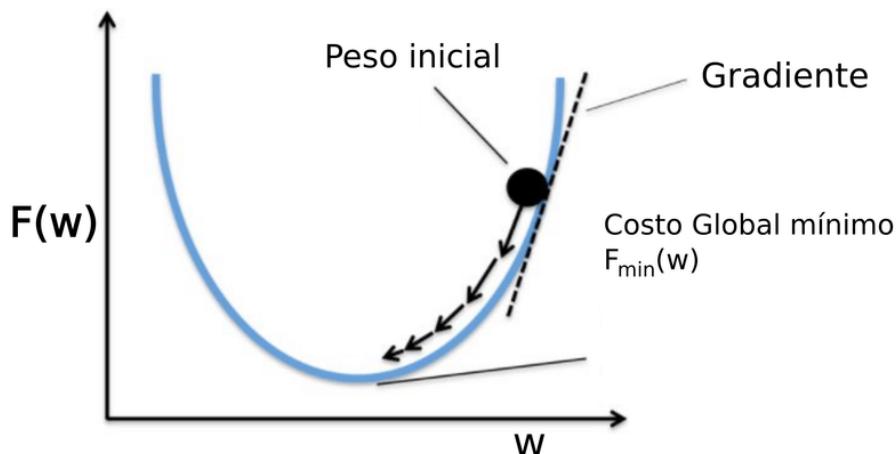


Figura 49: Representación gráfica de la dirección del gradiente³⁶

A esta técnica se le conoce como descenso por gradiente y es muy utilizada por los principales algoritmos optimizadores.

- **Stochastic Gradient Descent (SGD):** Introducido por Herbert y Sutton en 1951 [16], el descenso por gradiente estocástico constituye una variación del descenso por gradiente original. En esta variante, se procede a la actualización de cada uno de los pesos de la red para cada uno de los ejemplos, en función de los errores cometidos:

$$\theta_{nuevo} = \theta_{antiguo} - \alpha \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)}) \quad (17)$$

³⁶ [https://www.researchgate.net/profile/Juan-Vasquez-Gomez/publication/329453137/figure/fig10/\[...\]](https://www.researchgate.net/profile/Juan-Vasquez-Gomez/publication/329453137/figure/fig10/[...])

Este algoritmo acelera el entrenamiento ya que se incrementa la actualización de los pesos y, por otro lado, uno de los principales inconvenientes es la actualización de todos y cada uno de los pesos por cada uno de los ejemplos, lo que eleva el cómputo.

- **SGD with Momentum:** Con el paso del tiempo, Boris T. Polyak propuso su metodología, denominada Polyak's Heavy Ball Method [17], la cual planteaba la incorporación de un factor de “memoria” con el objetivo de optimizar la convergencia en algoritmos iterativos. La implementación de un término de memoria en el Stochastic Gradient Descent constituye una optimización que propicia la atenuación de las actualizaciones del gradiente, con el propósito de fomentar la convergencia y acelerar el proceso de optimización. Este refinamiento se erige como un imperativo para mitigar oscilaciones superfluas y potenciar la celeridad en el ámbito del aprendizaje profundo. En contraste con la actualización de los parámetros mediante el gradiente actual, Momentum almacena una *velocidad* acumulada del gradiente pasado, permitiendo que la optimización se desenvuelva de manera más fluida y eficiente. Se introduce una variable v_t que representa la acumulación del gradiente en iteraciones previas y se actualiza según la siguiente fórmula:

$$\begin{aligned} v_t &= \beta v_{t-1} + (1 - \beta) \nabla_{\theta} J(\theta) \\ \theta_{\text{nuevo}} &= \theta_{\text{antiguo}} - \alpha v_t \end{aligned} \quad (18)$$

- **Adagrad:** El algoritmo Adagrad, en su búsqueda de optimizar el algoritmo SGD, implementa un learning rate adaptativo para cada peso de la red, en lugar de emplear un único valor de tasa de aprendizaje para todos los parámetros. Adagrad ajusta de manera individualizada cada parámetro en función del gradiente acumulado. La premisa fundamental que subyace en Adagrad radica en que los pesos que exhiben mayores oscilaciones deben recibir una tasa de aprendizaje reducida, mientras que aquellos con variaciones más pequeñas deben mantener una tasa de aprendizaje elevada. De esta manera, el algoritmo se centra en ajustar los pesos más relevantes y minimizar la influencia de aquellos que no contribuyen de manera estable al descenso del gradiente.

$$\theta_{\text{nuevo}} = \theta_{\text{antiguo}} - \frac{\alpha}{\sqrt{G_{\text{antiguo}} + \epsilon}} \nabla_{\theta} L(\theta_{\text{antiguo}}) \quad (19)$$

Donde α es la tasa de aprendizaje inicial, G_{antiguo} es la suma acumulada de los cuadrados de los gradientes de la iteración previa, ϵ es un pequeño valor para evitar la división por cero.

- **RMSprop:** Este algoritmo se presenta como una solución al problema de Adagrad, mediante la adaptación del learning rate de cada peso utilizando la media móvil exponencial del gradiente. En contraste con la simple acumulación de gradientes de Adagrad, la media móvil exponencial permite que los gradientes más antiguos se “olviden”, lo que impide una parada prematura del entrenamiento. La regla de actualización empleada por este algoritmo se presenta a continuación:

$$\begin{aligned} v_t &= \rho v_{t-1} + (1 - \rho) [\nabla_\theta L(\theta)]^2 \\ \theta_{nuevo} &= \theta_{antiguo} - \frac{\alpha}{v_t + \epsilon} \nabla_\theta L(\theta) \end{aligned} \tag{20}$$

Donde ρ es el *decay rate* y ϵ es un pequeño valor para evitar la división por cero.

- **Adam:** Adaptative Moment Estimation, es un algoritmo presentado en 2014 por Diederik Kingma [18], que calcula un learning rate adaptativo para cada parámetro basándose en la media del primer y segundo momento del gradiente. Combina las ideas de SGD con Momentum y RMSprop, utilizando momentos de primer y segundo orden para ajustar dinámicamente la tasa de aprendizaje en cada parámetro. Este algoritmo requiere la elección de tres parámetros: el learning rate y los decay rates del primer (β_1) y segundo (β_2) momento. Utilizando estos parámetros, Adam ajusta el learning rate mediante los siguientes estimadores:

$$\begin{aligned} v_t &= \beta_1 * v_{t-1} - (1 - \beta_1) * g_t \\ s_t &= \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2 \\ \Delta w_t &= -\alpha \frac{v_t}{\sqrt{s_t + \Delta_\epsilon}} * g_t \\ w_{t+1} &= w_t + \Delta w_t \end{aligned} \tag{21}$$

Donde g_t representa el gradiente en w en el momento t .

B.2.5 Backpropagation

Tras el análisis de las limitaciones del perceptrón llevado a cabo por Minsky y Papert, las redes neuronales cayeron en el olvido durante un tiempo. Sin embargo, en la década de 1980 volvieron a cobrar relevancia gracias al trabajo de David E. Rumelhart y Geoffrey E. Hinton. Hinton y Ronald J. Williams [7], quienes demostraron cómo utilizar el algoritmo de propagación hacia atrás para entrenar redes neuronales con múltiples capas.

Este algoritmo se fundamenta en la regla de la cadena, un procedimiento empleado para calcular la derivada de una composición de funciones. Mediante este método, es factible determinar el gradiente de la función de pérdida con respecto a los pesos de las capas ocultas. A pesar de que no se cuente con el error en estas capas debido a la falta de una salida esperada, sí es posible identificar la influencia de cada peso en el error final de la red. Este proceso se logra mediante la descomposición del problema en una serie de operaciones encadenadas que permiten propagar el error desde la salida hacia las capas anteriores, ajustando los pesos de manera eficiente.

El algoritmo en cuestión se compone de dos fases principales, a saber, la propagación hacia adelante y la retropropagación del error. Mediante la implementación de estas dos fases, se logra el objetivo de minimizar la función de pérdida $J(\theta)$, mediante la optimización de los pesos W de

manera que la red produzca salidas que se aproximen más a los valores reales. A continuación, se procederá a un análisis detallado de estas fases.

En la primera fase, también conocida en inglés como *Forward Pass*, la información se transmite a través de la red neuronal desde la entrada hasta la salida, atravesando las capas ocultas. Inicialmente, se introduce una entrada X en la red y cada neurona en las capas subsiguientes calcula su activación en función de los pesos W y los sesgos b actuales. La activación en cada capa l se obtiene mediante la ecuación:

$$\begin{aligned} z^{(l)} &= W^{(l)}a^{(l-1)} + b^{(l)} \\ a^{(l)} &= \sigma(z^{(l)}) \end{aligned} \tag{22}$$

Donde σ es la función de activación (como por ejemplo ReLU, sigmoide, etc) que introduce la no linealidad en la red. Este proceso se repite capa por capa hasta alcanzar la capa de salida, donde se genera una predicción \hat{Y} basada en los pesos actuales. Posteriormente se realiza una comparación entre la predicción \hat{Y} y la salida real Y mediante una función de pérdida $J(\theta)$, que mide la eficacia de la red en su tarea. Esta función de pérdida será la que se use en la siguiente fase para ajustar los pesos y mejorar la precisión del modelo.

En la segunda fase, denominada en inglés *Backward Pass*, el propósito radica en ajustar los pesos de la red neuronal con el fin de minimizar la incertidumbre asociada a las predicciones. Este procedimiento se inicia en la capa de salida, donde se calcula la disparidad entre la salida esperada Y y la salida predicha \hat{Y} , generando un error que se manifiesta en términos del gradiente de la función de pérdida:

$$\delta^{(L)} = \frac{\partial J}{\partial a^{(L)}} * \sigma'(z^{(L)}) \tag{23}$$

En consecuencia, el error se propaga hacia atrás a través de las capas de la red utilizando la regla de la cadena, lo que permite distribuir la contribución del error a cada peso en las capas anteriores:

$$\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} * \sigma'(z^{(l)}) \tag{24}$$

Con estos valores de error en cada capa, se calculan los gradientes respecto a los pesos, siendo $\delta^{(l)}$ el error de cada capa:

$$\frac{\partial J}{\partial W^{(l)}} = \delta^{(l)} a^{(l-1)T} \tag{25}$$

Finalmente, los pesos de la red se actualizan utilizando el método de descenso por gradiente, ajustándolos en la dirección opuesta al gradiente para minimizar la función de pérdida, donde α representa la tasa de aprendizaje:

$$W^{(l)} = W^{(l)} - \alpha \frac{\partial J}{\partial W^{(l)}} \quad (26)$$

Este proceso de retropropagación se reitera en cada iteración del entrenamiento hasta que la función de pérdida alcanza un valor óptimo, permitiendo que la red aprenda representaciones más precisas para la tarea que se pretende resolver.

B.3 Hiperparámetros

El entrenamiento de una red neuronal se fundamenta en la selección meticulosa de diversos parámetros, los cuales deben ser escrupulosamente definidos con anterioridad al inicio del proceso. La arquitectura de la red, learning rate y batch size , constituyen meramente una muestra de las múltiples decisiones que deben ser tomadas de manera precisa y anticipada.

Estos valores se denominan hiperparámetros, puesto que no son adquiridos por la red, sino que establecen las condiciones en las cuales se efectuará el entrenamiento. Su configuración incide directamente en la capacidad del modelo para hallar los valores óptimos de los parámetros (pesos y sesgos) que permitirán a la red realizar predicciones con precisión.

La selección de los hiperparámetros se fundamenta en múltiples criterios, tales como la experiencia previa, el examen de enfoques empleados en el estado de la técnica para problemas afines y la ejecución de pruebas preliminares sobre el conjunto de datos de entrenamiento. El proceso de ajuste de hiperparámetros tiende a llevarse a cabo de manera iterativa, mediante la realización de múltiples experimentos en los que se buscan mejoras progresivas respecto a configuraciones previas. Esto puede implicar la exploración de nuevas combinaciones de hiperparámetros o un enfoque en la recolección y mejora de los datos, garantizando que el modelo tenga información más representativa y de mayor calidad para su aprendizaje.

A continuación, exponemos los principales hiperparámetros involucrados en el entrenamiento de una red, junto con algunas recomendaciones para su definición.

B.3.1 Arquitectura de red

La arquitectura de una red neuronal determina su capacidad de aprendizaje y debe estar alineada con la complejidad del problema que se busca resolver. Una estrategia común es utilizar una arquitectura previamente probada en problemas similares y con resultados contrastados, lo que puede ahorrar tiempo y esfuerzo en la configuración del modelo. Sin embargo, según el teorema No Free Lunch [15], no existe un modelo universalmente superior para todos los problemas, en otras palabras, si un algoritmo funciona bien para un conjunto específico de problemas,

inevitablemente tendrá un rendimiento deficiente en otros.
IE En consecuencia, la selección de la arquitectura debe fundamentarse en pruebas empíricas, ajustes iterativos y una comprensión profunda de las características del problema en cuestión.

B.3.2 Técnica de inicialización de los pesos

La inicialización de los pesos en una red neuronal constituye un hiperparámetro crítico que ejerce una influencia significativa en la velocidad y estabilidad del entrenamiento. Diversas estrategias han sido propuestas para definir estos valores antes de iniciar el proceso de ajuste de los parámetros. Se recomienda, en la medida de lo posible, recurrir a la técnica de transfer learning, mediante la reutilización de los pesos de una red previamente entrenada en un problema de naturaleza similar. Esta práctica permite iniciar el proceso de entrenamiento desde una configuración más cercana al óptimo, lo que se traduce en una reducción del tiempo de entrenamiento y un aumento de la probabilidad de alcanzar una convergencia más efectiva. En ausencia de un modelo preentrenado, es posible recurrir al uso de métodos de inicialización, como los mencionados en el Anexo B en su apartado B.2.3 Inicialización de los pesos

, los cuales contribuyen a mejorar la propagación de la señal a lo largo de la red y evitan problemas como la saturación de las funciones de activación. En contraste, una inicialización completamente aleatoria puede ocasionar un aumento en el tiempo de entrenamiento o incluso impedir que el modelo converja de manera apropiada.

B.3.3 Número de épocas

El entrenamiento de una red neuronal se estructura en un número determinado de épocas, cada una de las cuales representa un ciclo completo en el que el modelo procesa todo el conjunto de datos de entrenamiento.

En el caso de que el número de épocas sea insuficiente, es posible que la red no disponga de un tiempo suficiente para ajustar sus pesos, lo que puede resultar en un modelo con un alto error y un rendimiento deficiente, lo que se conoce como underfitting. Por otro lado, un número excesivo de épocas puede conducir al sobreajuste, denominado en inglés overfitting, en el que el modelo aprende de forma excesiva los datos de entrenamiento, pero pierde capacidad de generalización en datos nuevos, como se observa en la Figura 50.

El número óptimo de épocas es un aspecto crucial que debe determinarse en función de diversos factores. Estos factores incluyen la complejidad del problema, el tamaño del conjunto de datos y la arquitectura de la red. Para evitar el sobreentrenamiento, se recomienda el uso de técnicas como early stopping, que detiene el entrenamiento cuando el rendimiento en el conjunto de validación deja de mejorar.

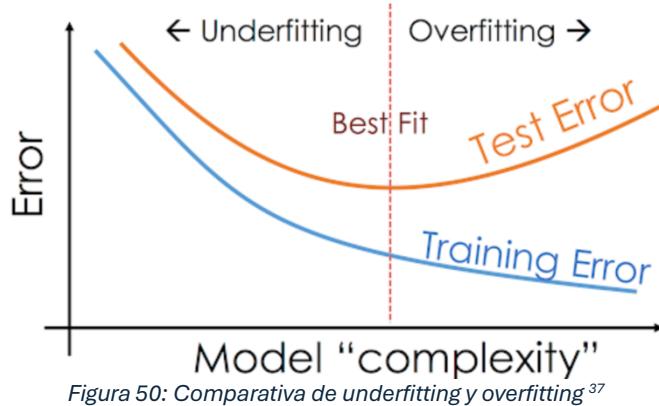


Figura 50: Comparativa de underfitting y overfitting³⁷

B.3.4 Batch Size

En lo que respecta al tamaño del lote de ejemplos, es pertinente señalar que la cantidad de datos empleados en el cálculo del valor de pérdida ejerce una influencia significativa en la configuración de la superficie de la función de pérdida y, por consiguiente, en la complejidad que enfrentará el algoritmo de optimización al intentar alcanzar el mínimo. En este contexto, se hace común el uso de potencias de dos para definir el tamaño del lote, siendo los valores 16, 32, 64 y 128 los más prevalentes.

B.3.5 Almacenamiento de los pesos

El almacenamiento de los pesos de la red constituye un aspecto de vital importancia en el marco del entrenamiento, toda vez que el propósito del experimento radica en la obtención de los pesos óptimos para la problemática en cuestión. La práctica más aconsejable consiste en el almacenamiento de los pesos en el momento en el que se evidencie una mejora en la función de pérdida o en alguna métrica de validación, tales como la precisión o el AUC-ROC. Esta práctica garantiza que los parámetros almacenados correspondan al mejor desempeño alcanzado durante el entrenamiento. Para gestionar este proceso, se suelen emplear estrategias como el early stopping con checkpointing, donde se guarda el mejor modelo en función de un criterio específico y se evita el sobreentrenamiento.

B.3.6 Algoritmo de optimización

El algoritmo de optimización determina la forma en que se emplean las derivadas obtenidas mediante el método de backpropagation para actualizar los pesos de la red neuronal. Si bien backpropagation constituye el único método para calcular los gradientes en las capas ocultas, la manera en que se aplican estos gradientes depende del optimizador seleccionado.

Algunos algoritmos, como el descenso de gradiente, son más rápidos de calcular en comparación con otros más avanzados, como Adam. Una de las razones por las que se opta por el descenso de gradiente es su simplicidad, lo que facilita el entrenamiento y reduce el riesgo de

³⁷ <https://cdn.analyticsvidhya.com/wp-content/uploads/2020/02/...>

sobreajuste. Además, en el caso de que el algoritmo llegue a un punto de equilibrio, es probable que este sea un mínimo aplanado en lugar de un mínimo pronunciado, lo que aumenta las posibilidades de que se haya encontrado el mínimo global y no un mínimo local, como se muestra en la Figura 51.

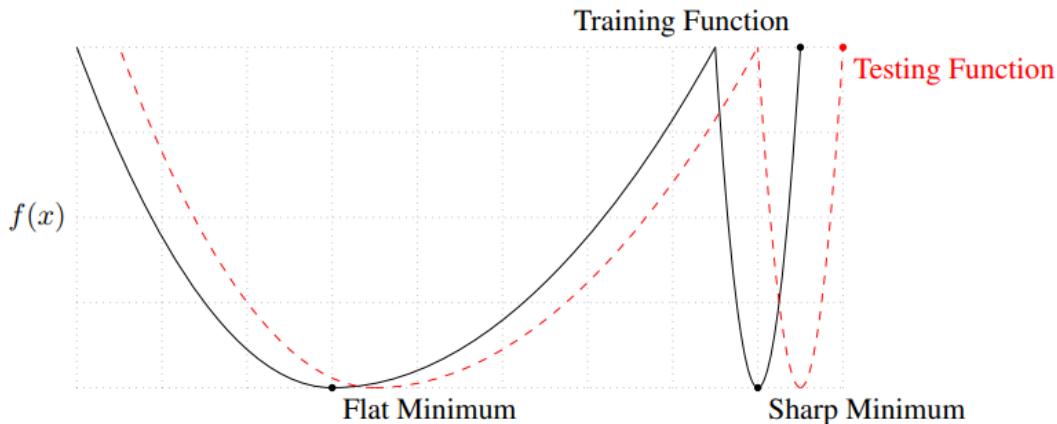


Figura 51: Diferencia gráfica entre mínimo aplanado y mínimo profundo³⁸

No obstante, se ha comprobado que optimizadores como Adam, RMSprop o Adagrad incluyen mejoras como tasas de aprendizaje adaptativas y momentum, lo que permite una convergencia más rápida y estable en problemas con funciones de pérdida más. Por tanto, se puede concluir que la elección del algoritmo de optimización dependerá del problema, el tamaño del conjunto de datos y la arquitectura de la red.

Entre los parámetros del algoritmo, el learning rate es uno de los más críticos, no solo en el contexto del algoritmo de optimización, sino también en el proceso global de entrenamiento. Este parámetro determina la magnitud de los ajustes efectuados en los pesos de la red durante cada iteración. Un learning rate excesivamente alto puede provocar que el modelo no converja o incluso diverja, oscilando sin alcanzar un mínimo. Por otro lado, un learning rate excesivamente bajo puede ocasionar que el entrenamiento se vuelva excesivamente lento y quede atrapado en mínimos locales (ver Figura 52).

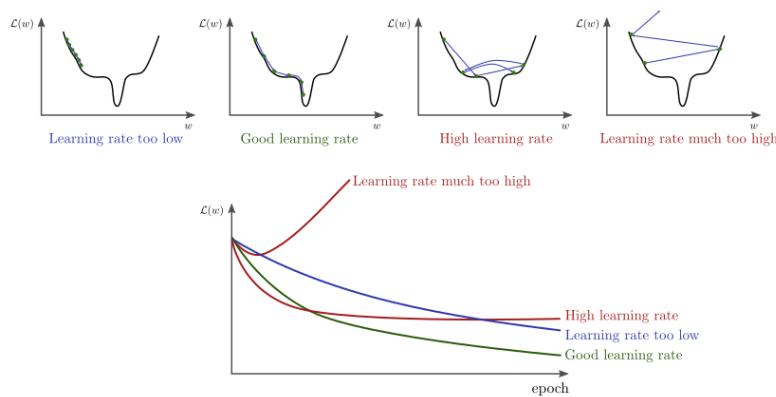


Figura 52: Efecto de diferentes tasas de aprendizaje en un entrenamiento³⁹

³⁸ <https://imgur.com/hgGxHZe>

³⁹ <https://i0.wp.com/spotintelligence.com/wp-content/uploads/2024/02/...>

B.3.7 Métricas

En la evaluación de modelos, se observa la existencia de una diversidad de métricas diseñadas para satisfacer los requerimientos específicos de diversos tipos de problemas. La selección de dichas métricas debe estar en consonancia con los objetivos del modelo, ya que son estas métricas las que determinan su rendimiento y, simultáneamente, sirven como criterio para la determinación del momento oportuno para almacenar los pesos que posteriormente serán empleados en la fase de inferencia.

En problemas de clasificación, se suelen emplear métricas como la precisión, el recall, la F1-score y el AUC-ROC, mientras que, en regresión, métricas como el error cuadrático medio, el error absoluto medio o el coeficiente de determinación son más adecuadas.

La elección de las métricas de evaluación también depende del tipo de problema que se está abordando, en tanto que, en ciertos escenarios, algunas métricas pueden adquirir una relevancia superior a otras. A modo ilustrativo, en el ámbito de la detección de enfermedades, es fundamental minimizar los falsos negativos, habida cuenta de que una predicción errónea que clasifique a un paciente sano como enfermo podría acarrear consecuencias fatales. En contraste, en problemas como la detección de fraudes financieros, se busca minimizar los falsos positivos para evitar que transacciones legítimas sean erróneamente clasificadas como fraudulentas.

Por lo tanto, al seleccionar las métricas, no solo se debe considerar el rendimiento general del modelo, sino también el impacto que diferentes tipos de errores pueden tener en el contexto específico del problema.

B.3.8 Función de pérdida

La función de pérdida constituye un componente esencial en el entrenamiento de una red neuronal. Su rol radica en cuantificar la discrepancia entre la salida predicha por el modelo y el valor esperado, permitiendo la optimización de los parámetros de la red. En lo que respecta a la selección de la función de pérdida, se permite la utilización de cualquier función derivable, garantizando así la capacidad del algoritmo de backpropagation para calcular los gradientes y actualizar los pesos de manera eficiente.

La elección de la función de pérdida se determina en función del tipo de problema que se busca resolver. En el caso de la regresión, se emplean comúnmente funciones como el error cuadrático medio (MSE) o el error absoluto medio (MAE), mientras que en la clasificación se utilizan funciones como la entropía cruzada (Cross-Entropy Loss) o su variante binaria (Binary Cross-Entropy, BCE). En problemas más específicos, es posible diseñar funciones de pérdida personalizadas que penalicen ciertos errores de manera diferente según su impacto en la tarea a resolver.

Una vez que se han establecido los hiperparámetros, se procederá a la realización de un experimento, y en función de las métricas obtenidas, se determinará la siguiente prueba a ejecutar. Este proceso iterativo permite ajustar el modelo progresivamente hasta alcanzar un rendimiento óptimo.

En la siguiente sección, se presentan algunas directrices para garantizar un entrenamiento exitoso e identificar posibles problemas que puedan surgir durante el proceso. La identificación de estos problemas permitirá optimizar la selección de los hiperparámetros, comprender qué parámetros requieren ser modificados y cómo, por ende, mejorar el desempeño del modelo de manera más eficiente.

B.4 Control y seguimiento

Es habitual llevar un registro de los valores de pérdida y de las métricas a lo largo del proceso de entrenamiento, con el propósito de realizar un análisis de su evolución. En este sentido, se almacena los valores de pérdida en cada iteración y, cuando se considera pertinente, las métricas calculadas en cada paso. Asimismo, al concluir cada época, se almacenan los valores promedio tanto para el conjunto de entrenamiento como para el conjunto de validación.

Durante el proceso de entrenamiento, se anticipa una disminución progresiva en los valores de pérdida y una mejora en las métricas de rendimiento. Además, se espera que los resultados obtenidos en el conjunto de entrenamiento sean similares a los del conjunto de validación. En caso de que se observen comportamientos anómalos, tales como una falta de mejora o una divergencia significativa entre ambas curvas, esto puede indicar problemas como sobreajuste, subajuste o una mala configuración de los hiperparámetros, lo que requerirá ajustes en el modelo o en la estrategia de entrenamiento.

El análisis de los valores de pérdida y métricas se lleva a cabo mediante una representación gráfica en la que el eje X representa las iteraciones o épocas del entrenamiento, mientras que el eje Y muestra el valor de pérdida o la métrica seleccionada, como se muestran en Figura 53.

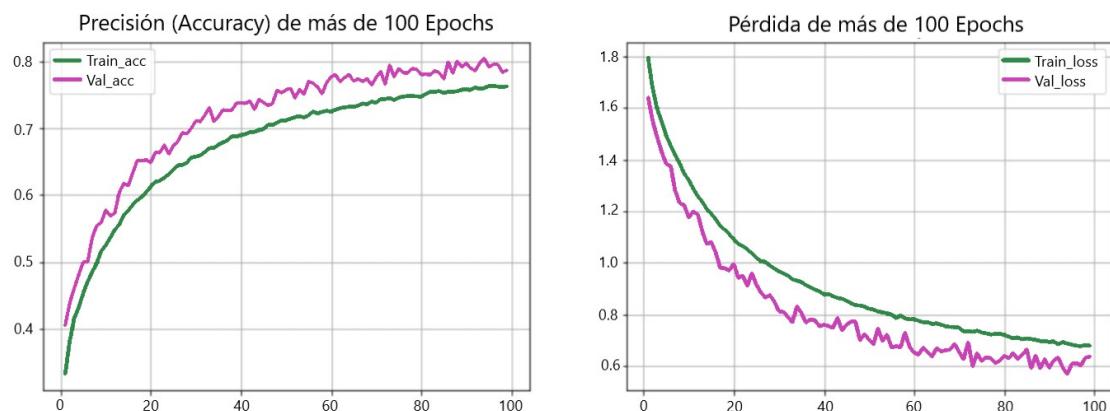


Figura 53: Ejemplo de gráficas de loss y accuracy en un entrenamiento de 100 épocas⁴⁰

⁴⁰ https://sitiobigdata.com/wp-content/uploads/2023/09/precision_perdida_epoch.jpg

En los estadios iniciales del entrenamiento, el valor de pérdida tiende a ser elevado debido a que los pesos de la red han sido asignados de acuerdo con la técnica de inicialización seleccionada. No obstante, conforme progresá el entrenamiento y el algoritmo de optimización ajusta los parámetros, la pérdida comienza a disminuir de manera significativa tras un determinado número de épocas.

Se observa que las métricas de evaluación exhiben una tendencia a iniciar con valores bajos y a presentar un incremento progresivo a medida que se reduce la pérdida. Este fenómeno se explica por la correlación entre la reducción de la pérdida y la capacidad del modelo para realizar predicciones más precisas, lo que se manifiesta en un aumento de las métricas de precisión y recall.

Se espera que la disminución de la pérdida esté asociada con la optimización de las métricas, dado que ambas reflejan la eficacia del modelo. Sin embargo, si se evidencia que la pérdida continúa reduciéndose mientras las métricas se estabilizan o muestran un deterioro.

El análisis de las gráficas de pérdida y métricas permite detectar la mayoría de los problemas que pueden surgir durante el entrenamiento, como el sobreajuste o el infraajuste. En el caso de que la pérdida registrada en el conjunto de entrenamiento sea reducida, mientras que la pérdida en el conjunto de validación se mantenga en niveles elevados, se puede inferir que el modelo se encuentra sobreajustando los datos de entrenamiento, lo que sugiere una incapacidad para generalizar a nuevos datos. En esta situación, la red memoriza el conjunto de entrenamiento en lugar de desarrollar habilidades para aprender patrones generales, lo que resulta en un rendimiento deficiente en lo que respecta a la validación. Para corregir este sobreajuste, se han propuesto diversas estrategias, tales como el incremento del tamaño del conjunto de datos, la reducción de la complejidad del modelo mediante la eliminación de capas o neuronas, y la implementación de técnicas de regularización como dropout o L2.

En contraste, cuando existe una correlación entre la pérdida del conjunto de entrenamiento y la de validación, ambas permanecen en niveles elevados, se evidencia un subajuste en el modelo. Este fenómeno sugiere una limitación en la capacidad de la red para asimilar adecuadamente los patrones asociados al problema, lo que impide una reducción significativa del error. En tal caso, la solución propuesta consiste en incrementar la complejidad del modelo mediante la incorporación de capas o neuronas adicionales. Esta medida se anticipa que permitirá a la red exhibir una capacidad de representación ampliada, lo que a su vez facilitará la reducción continua del error.

Una vez identificado un problema en el entrenamiento de la red neuronal, es posible implementar diversas técnicas para corregirlo y optimizar la convergencia del modelo. A continuación, se presentan algunas de las estrategias más comúnmente utilizadas:

- **Equilibrar el número de muestras de las clases:** El equilibrio en el número de ejemplos por clase es una estrategia clave para evitar problemas de sobreajuste cuando el conjunto de datos está desbalanceado. No siempre el sobreajuste es causado por un modelo demasiado grande, sino que también puede deberse a que una clase es predominante dentro del conjunto de entrenamiento.

Para evitar este problema, se acude a varias técnicas. En el proceso de validación de datos, es crucial mantener un equilibrio entre los ejemplos presentados, garantizando que el conjunto de datos de validación contenga una cantidad uniforme de ejemplos por cada categoría o clase de datos. Este equilibrio es fundamental para asegurar la precisión y representatividad de las métricas obtenidas durante el proceso de validación. En lo que respecta a la distribución de las clases en los lotes de entrenamiento, se ha observado que la implementación de un balanceo en los lotes resulta en una distribución más uniforme de las clases dentro de cada lote. Este procedimiento se ha implementado con el propósito de evitar que la red aprenda patrones erróneos basados en la frecuencia de aparición de una clase particular. Otras de las técnicas más utilizadas es la de remuestreo, se pueden aplicar métodos como el submuestreo de la clase mayoritaria o el sobremuestreo de la clase minoritaria para balancear el conjunto de entrenamiento.

- **Aplicar criterios de actualización del learning rate:** En los algoritmos de optimización que no emplean un learning rate adaptativo, es común aplicar políticas de actualización del learning rate para evitar que este mantenga un valor constante durante todo el proceso de entrenamiento. Al inicio del entrenamiento, un learning rate elevado permite que el modelo descienda rápidamente en la función de pérdida, acercándose más rápidamente al mínimo.

No obstante, en etapas posteriores, este valor puede volverse problemático, impidiendo que el modelo refine su ajuste si los pasos de actualización son demasiado grandes. Para evitar este problema, se pueden utilizar diferentes estrategias para actualizar el learning rate. Una de ellas es la disminución por épocas fijas, que consiste en reducir el valor del learning rate después de un cierto número de épocas predefinidas.

En segundo lugar, se encuentra la reducción basada en la pérdida, que se aplica cuando el valor de la pérdida se mantiene estable durante varias épocas, lo que permite ajustes más finos.

Por último, se puede emplear la decay exponencial, que se caracteriza por usar una función exponencial para reducir progresivamente el learning rate a lo largo del entrenamiento.

Estas políticas permiten optimizar el proceso de aprendizaje, asegurando una convergencia estable y evitando que el modelo quede atrapado en un mínimo local o no termine de ajustar correctamente los pesos.

- **Normalización:** la normalización de los datos constituye un paso fundamental en el proceso de preprocesamiento, ya que facilita la optimización de la red neuronal y garantiza su estabilidad. En este sentido, cuando los datos poseen una media de cero y una desviación típica de uno, el modelo puede alcanzar una capacidad de aprendizaje más eficiente, ya que todas las dimensiones se encuentran en un rango similar.

Sin embargo, si alguna de las dimensiones de entrada presenta valores significativamente superiores en comparación con las demás, la superficie de pérdida puede experimentar una elongación, lo que complica la optimización y genera oscilaciones en el entrenamiento. Estas oscilaciones pueden dificultar el proceso de descenso del algoritmo de optimización hacia el mínimo óptimo de la función de pérdida.

- **Regularización:** La regularización constituye una metodología empleada para mitigar la complejidad del modelo y prevenir el sobreajuste. Este procedimiento implica la incorporación de una penalización en el cálculo del error, la cual está determinada en función del valor de los pesos de la red neuronal. Los modelos de mayor complejidad y propensos al sobreajuste tienden a exhibir valores elevados en sus pesos, lo que resulta en la generación de cambios bruscos en la salida a partir de pequeñas variaciones en la entrada. La regularización, por tanto, se presenta como un mecanismo que mitiga el efecto anteriormente descrito, promoviendo la reducción de los valores de los pesos y, por ende, suavizando la función de decisión y potenciando la capacidad de generalización del modelo.
- **Batch normalization:** es una técnica que permite la normalización de las activaciones de las distintas capas de la red neuronal, en contraposición a la normalización de los datos de entrada. A medida que los datos se propagan a través de las capas de la red, sus distribuciones pueden experimentar cambios significativos, lo que puede resultar en valores extremos y dificultar la convergencia del modelo.

Para abordar esta problemática, se implementa el ajuste de la distribución de las activaciones en cada capa mediante la normalización de los valores dentro de cada mini-lote, lo que implica la resta de la media y la división por la desviación estándar del lote actual de ejemplos. Posteriormente, se aplican dos parámetros adicionales de escalado y desplazamiento para facilitar la adquisición de distribuciones más flexibles, en caso de ser requerido.

- **Dropout:** su propósito es evitar el sobreajuste en dichos sistemas. Su funcionamiento se fundamenta en la desactivación aleatoria de un conjunto de neuronas pertenecientes a una capa específica durante el proceso de entrenamiento. Esta estrategia transforma la red en un sistema compuesto por múltiples redes de menor complejidad, las cuales aprenden a cooperar para resolver el problema en cuestión, promediando sus salidas. En el caso de que Dropout se aplique en una capa con n neuronas, cada entrenamiento se realizará sobre una versión reducida de la red, lo que equivale a entrenar hasta 2^n subredes diferentes. En cada iteración, se seleccionará una de estas subredes y se ajustarán los pesos solo de las neuronas activas.
- **Data augmentation:** este enfoque de mejora de la generalización de un modelo sin la necesidad de modificar su arquitectura ni aplicar técnicas adicionales de regularización. Este enfoque implica la ampliación artificial del conjunto de datos, una estrategia particularmente beneficiosa cuando se cuenta con un número limitado de muestras y se busca evitar el sobreajuste.

En el contexto de un problema de clasificación de imágenes médicas, donde el conjunto de datos proviene de un grupo reducido de pacientes, existe la probabilidad de que la red neuronal termine memorizando las características específicas de dichas imágenes en

lugar de aprender patrones generales. Para abordar esta situación, se pueden aplicar diversas transformaciones a las imágenes originales, tales como rotaciones, volteos, recortes, escalados, variaciones en el brillo y contraste, o la adición de ruido aleatorio. La selección de técnicas de aumento de datos se determina en función del tipo de problema específico. En el ámbito del procesamiento de lenguaje natural, se pueden generar sinónimos, reorganizar frases o eliminar palabras irrelevantes para la tarea. En el campo del reconocimiento de voz, se puede modificar la velocidad o el tono de los audios. El objetivo principal es generar nuevas muestras a partir de las existentes que sean lo suficientemente distintas del dato original, pero que conserven su significado para que el modelo pueda seguir identificándolas.

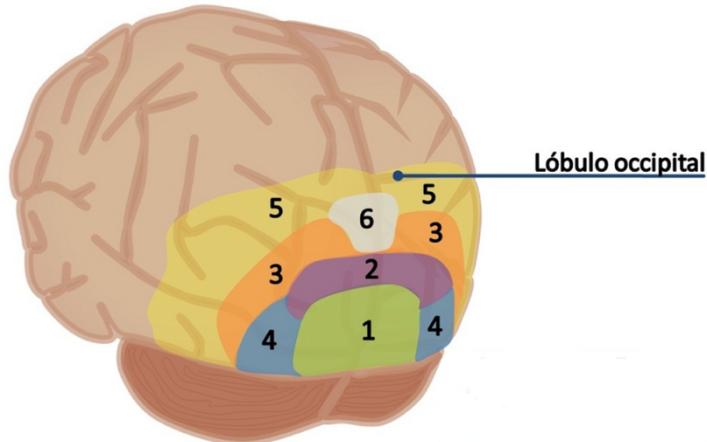
Anexo teórico C Redes neuronales convolucionales

C.1 Introducción

Como vimos en el capítulo anterior, las redes neuronales artificiales tienen una inspiración biológica en la estructura del cerebro humano. En el cerebro, millones de neuronas están interconectadas entre sí y se comunican mediante impulsos eléctricos; si una neurona recibe suficiente estímulo, se activa y genera una nueva señal hacia otras neuronas. Esta cadena de estímulos que da lugar a los comportamientos complejos del ser humano es lo que se intenta imitar con las redes neuronales artificiales.

Dentro de las redes neuronales encontramos diversas clases entre las cuales destacan las redes neuronales convolucionales (CNN). Como todas las redes neuronales, se basa en el cerebro humano, pero en este caso específicamente en el córtex visual. El córtex es responsable de percibir y reconocer elementos y formas que el ojo humano observa, comenzando con elementos simples como líneas o curvas (corteza visual primaria o región V1) y progresando a patrones más complejos como rostros (áreas visuales corticales extraestriadas o regiones V2, V4) como se muestra en la Figura 54.

Área encargada de la visión:



Lóbulo occipital: Lóbulo cerebral posterior dedicado fundamentalmente a la visión.

Áreas:

- 1.- Exploración e inscripción general.
- 2.- Visión estereoscópica.
- 3.- Profundidad y distancia.
- 4.- Color.
- 5.- Movimiento.
- 6.- Determinación de la posición absoluta del objeto.

Figura 54: Áreas del lóbulo cerebral posterior⁴¹

⁴¹ <https://blogger.googleusercontent.com/img/b/...>

Para replicar este comportamiento las CNNs utilizan diferentes tipos de capas más allá de las densamente conectadas de las redes tradicionales. Estas nuevas capas permiten procesar datos en formatos bidimensionales, lo cual supera las capacidades de las redes clásicas y permite abordar problemas relacionados con la visión artificial.

La principal diferencia entre las redes tradicionales y las convolucionales se encuentra en la capacidad que tienen estas últimas para manejar estructuras bidimensionales, en las clásicas las entradas se expresan en una única capa plana, lo que hacía inviable introducir una imagen directamente, ya que se perdía toda la información estructural y el número de entradas sería excesivo haciendo que el experto debía analizar las imágenes y extraer los descriptores clave, para alimentar la red. Con las CNNs esto no ocurre así, las imágenes se pueden recibir directamente como entrada eliminando la necesidad del análisis previo.

Esta extracción de características se efectúa a través de varias capas. Las capas de convolución son las encargadas de aprender a reconocer patrones en las imágenes, combinadas con otras operaciones y capas son capaces de detectar elementos estructurales cada vez más complejos a medida que la red se profundiza, desde la detección de bordes simples hasta formas elaboradas como rostros o gatos (ver Figura 55).

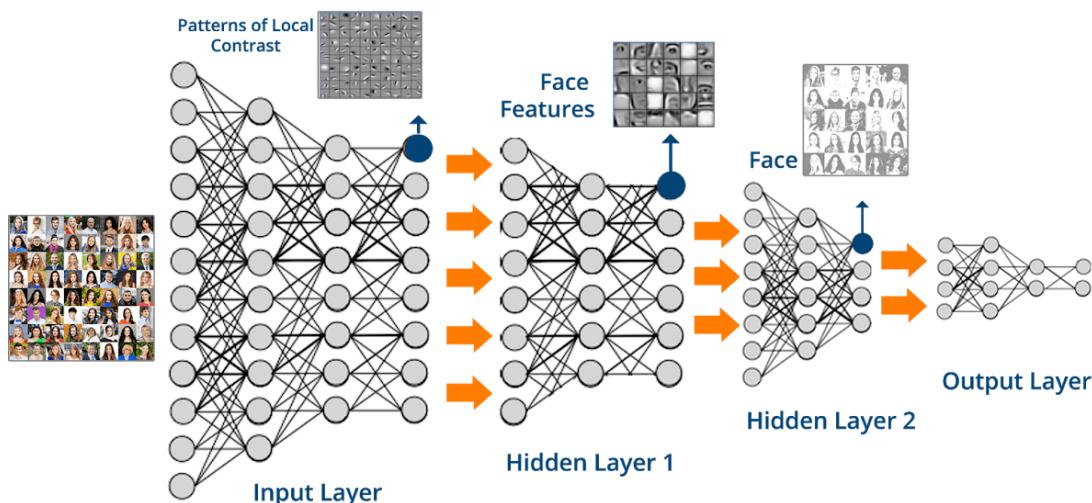


Figura 55: Ejemplo de extracción de características en capas de convolución⁴²

Entrenar redes convolucionales es similar a entrenar redes neuronales clásicas, pero con la diferencia de que también se ajustan los valores de los kernels de convolución. Las capas de convolución ayudan a la red a extraer características cada vez más representativas. Sin embargo, encontrar los valores óptimos para los filtros depende de la arquitectura de la red. Tener más capas y filtros permite reconocer características más complejas, pero también incrementa el número de parámetros a ajustar, lo que aumenta el tiempo de entrenamiento. Si hay demasiados parámetros y la red no está bien inicializada, es posible que el modelo no logre converger.

⁴² [https://4.bp.blogspot.com/-pUmPcAL5LEc/XbLLk_UzhNI/AAAAAAAAB8M/\[%\]](https://4.bp.blogspot.com/-pUmPcAL5LEc/XbLLk_UzhNI/AAAAAAAAB8M/[%])

Para entrenar una red neuronal convolucional es necesario realizar múltiples pruebas con diferentes estructuras para determinar el número óptimo de capas para el problema en cuestión. Además, es importante ajustar la cantidad y el tamaño de los filtros para encontrar un equilibrio entre eficiencia y eficacia. En el resultado final influyen la correcta inicialización de los parámetros, la elección del algoritmo de entrenamiento, las funciones de error a utilizar, ...

Es común aplicar una capa de pooling después de una operación de convolución para reducir el tamaño de los datos y mantener las características más relevantes, disminuyendo así el número de neuronas y parámetros y reduciendo significativamente el tiempo de entrenamiento.

C.2 Tipos de capas

Las redes neuronales convolucionales, como mencionamos anteriormente, están compuestas por una multitud de capas diferentes a las de las redes clásicas, que emplean otro tipo de capas. Estas capas permiten realizar una variedad de operaciones dentro de la red. Cada capa tiene características únicas y cumple una función específica. A continuación, describiremos algunas de las capas más utilizadas en las redes neuronales convolucionales y daremos ejemplos de su aplicación

C.2.1 Capas de convolución

Esta capa se encarga de extraer y transformar las características presentes en las matrices que recibe como entrada. Estas matrices pueden ser la imagen original o la salida de una capa previa de la red. Dado que el objetivo principal de esta capa es procesar las características de los datos, denominaremos a sus entradas y salidas como features maps cuando estas provengan de una capa previa o se dirigen a una capa siguiente.

El procedimiento mediante el cual esta capa identifica y procesa las características de los datos es aplicando filtros o kernels diseñados para el reconocimiento de diferentes patrones y formas en los features maps (ver Figura 56).

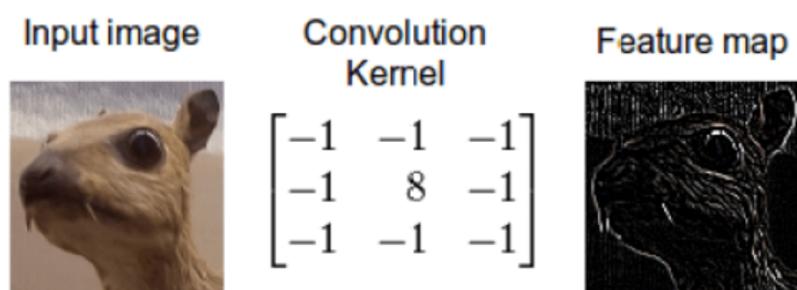


Figura 56: Detección de bordes sobre una imagen mediante convolución⁴³

La convolución es una operación matemática simple que involucra dos matrices de entrada generalmente de tamaños diferentes, pero con el mismo número de dimensiones, que se

⁴³ <https://i0.wp.com/timdettmers.com/wp-content/uploads/2015/03/...>

combinan para generar una tercera matriz. Estas matrices de entrada pueden ser los mapas de características recibidos de una capa previa o la propia imagen original, y el kernel pertenece a la capa de convolución actual.

El proceso de convolución consiste en deslizar el filtro o kernel sobre todas las posiciones del mapa de características de entrada donde dicho filtro encaje. Es decir, todos los valores de la matriz del filtro se superponen a un valor del mapa de características. Por cada posición en la que se sitúa el filtro sobre la matriz de entrada, se realiza una suma del producto de cada elemento del filtro por el valor subyacente de la matriz sobre la que está situado. El resultado de esta operación se almacena en un nuevo valor en el mapa de características de salida, como se puede apreciar en la Figura 57.

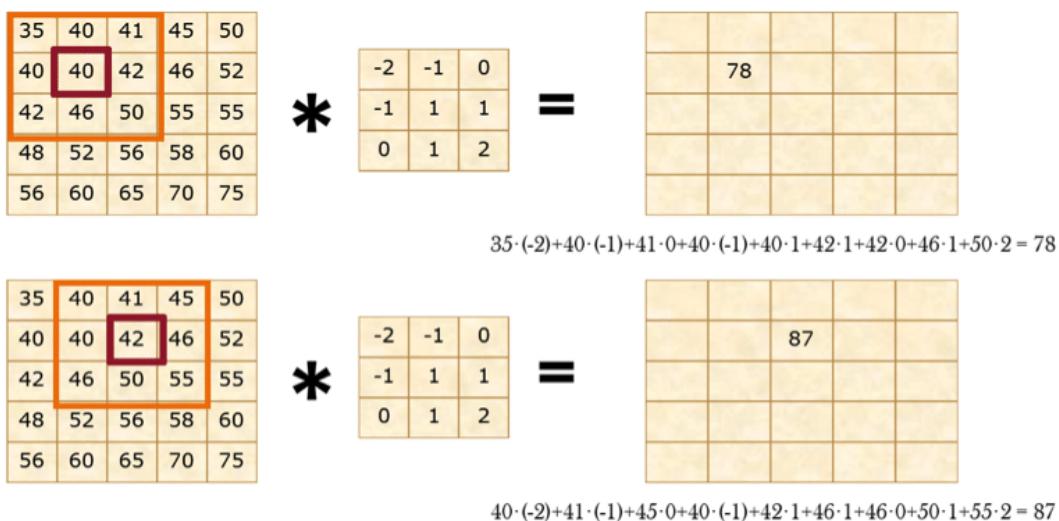


Figura 57: Operación de convolución⁴⁴

Esta capa es fundamental porque permite a la red neuronal convolucional extraer y reconocer patrones complejos en los datos de entrada, lo que es esencial para tareas como el reconocimiento de imágenes y la detección de objetos.

Matemáticamente la operación de convolución de aplicar un filtro K bidimensional de tamaño $m \times n$ sobre una imagen I también bidimensional de tamaño $M \times N$, se define con el operador * como:

$$(I * K)(i, j) = \sum_{k=1}^m \sum_{l=1}^n I(i+k-1, j+l-1) \cdot K(k, l) \quad (27)$$

donde i y j son las posiciones de la imagen resultante y recorren todas las filas y columnas respectivamente de la imagen I en donde el filtro K puede superponerse. En otras palabras, el resultado del filtro está desplazado.

⁴⁴ [https://www.researchgate.net/publication/292187589/figure/fig1/\[...\]](https://www.researchgate.net/publication/292187589/figure/fig1/[...])

Existe además la posibilidad de agregar un término constante más a la operación, llamado *bias*. De esta forma se asemeja más a los modelos matemáticos originales de las neuronas, quedando la fórmula:

$$(I * K)(i, j) = \sum_{k=1}^m \sum_{l=1}^n I(i + k - 1, j + l - 1) \cdot K(k, l) + b \quad (28)$$

Durante el entrenamiento, la elección de los filtros de convolución se realiza de forma automática ya que la etapa de convolución se aplica a un conjunto de filtros permitiendo al algoritmo seleccionar las características más relevantes.

Aunque estos filtros parezcan no tener sentido para la percepción humana, para la máquina pueden ser descriptores clave. Sin embargo, algunos parámetros importantes de la capa de convolución, como el tamaño inicial de la entrada de las imágenes, el tamaño de los kernels y la forma en que se aplican son definidos por el experto que desarrolla la red. Estos parámetros son cruciales para el rendimiento y la precisión del modelo. Vamos a explicarlos en más detalle:

- **Tamaño del filtro:** Es necesario indicar las dimensiones que tendrá el filtro de la convolución. Cuando mayor es el kernel, las características que se detecten podrán ser mayores.
- **Número de convoluciones:** Cantidad de mapas de características que se obtendrán como salida después de aplicar la capa de convolución actual. Cada operación de convolución recibe como entrada todos los mapas de características de la capa previa.
- **Padding:** Técnica utilizada para evitar la pérdida de píxeles en los extremos de una imagen. Normalmente al aplicar un kernel, los píxeles de los bordes pueden quedar fuera del alcance, lo que reduce el tamaño de la imagen resultante. El padding resuelve este problema ampliando los límites de la imagen original, de manera que el kernel pueda encajar en todas las posiciones, incluidos los bordes. Los píxeles añadidos durante el padding suelen rellenarse con ceros, replicar el valor del píxel más cercano o utilizar una forma de espejo.
- **Stride:** Define cómo se desliza el kernel sobre la matriz o imagen de entrada durante la convolución. Su valor determina el tamaño del "salto" que el kernel da mientras se desplaza. Por ejemplo, con un stride de 1, el kernel se aplica a cada elemento de la imagen, pero con un stride de 3, se aplicaría cada 3 elementos. Esto significa que genera una salida con un tamaño aproximadamente igual a un tercio del tamaño de la entrada.

La combinación de parámetros como el tamaño de padding, el kernel y el stride que escojamos influye directamente en la dimensión de la imagen de salida y en el tamaño de entrada a la siguiente capa de la red.

Por lo tanto, estos parámetros deben elegirse con cuidado. Para calcular las dimensiones de los mapas de características resultantes después de aplicar una operación de convolución, se puede utilizar la siguiente fórmula:

$$\text{Dimensión de salida} = \frac{(W - K + 2P)}{S} + 1 \quad (29)$$

donde W es el tamaño de entrada, K es el tamaño del *kernel*, S es el tamaño del *stride* y P es el tamaño del *padding*.

C.2.2 Capas de activación

Al igual que en las redes neuronales clásicas, en las redes neuronales convolucionales también se pueden utilizar estas funciones después de las capas de convolución. Esto añade una característica de no-linealidad ampliando la variedad de problemas que se pueden abordar al permitir que la red aprenda patrones más complejos.

La función de activación se aplica a cada elemento del mapa de características obtenido en la capa de convolución previa, alterando sus valores, pero no sus dimensiones. (ver Tabla 14: Principales funciones de activación).

C.2.3 Capas de pooling

Las capas de pooling o submuestreo, tienen como principal objetivo reducir las dimensiones de los mapas de características generados por una capa previa, generalmente una capa de convolución, sin perder las características más importantes presentes en dichos mapas. Esto permite disminuir el número de parámetros de la red y reduce el tiempo de entrenamiento y ejecución.

El pooling es especialmente útil en redes diseñadas para problemas de clasificación, donde es crucial condensar la información de las imágenes en tamaños muy reducidos, manteniendo únicamente los elementos que optimicen la separabilidad entre clases. Esto facilita la conexión a una capa final que actúe como clasificador.

Funcionalmente, la operación de pooling es similar a la convolución en el sentido de que ambas operan sobre la imagen de manera localizada, deslizando una ventana (equivalente al kernel en la convolución) sobre la imagen. Por lo tanto, el pooling comparte con la convolución parámetros como el tamaño del kernel y el stride.

A diferencia de la capa de convolución, en la operación de pooling el valor de salida no se calcula mediante una operación que involucre elementos propios de la capa actual, sino únicamente los valores del mapa de características. Esto convierte a la capa de pooling en una capa no entrenable de la red, cuyo comportamiento se define en el momento de construir la red. Las dos opciones más empleadas, mostradas en la Figura 58, son:

- **MaxPooling:** Se asigna el máximo valor de todos los valores de la ventana.
- **AveragePooling:** Se asigna la media de los valores de la ventana.

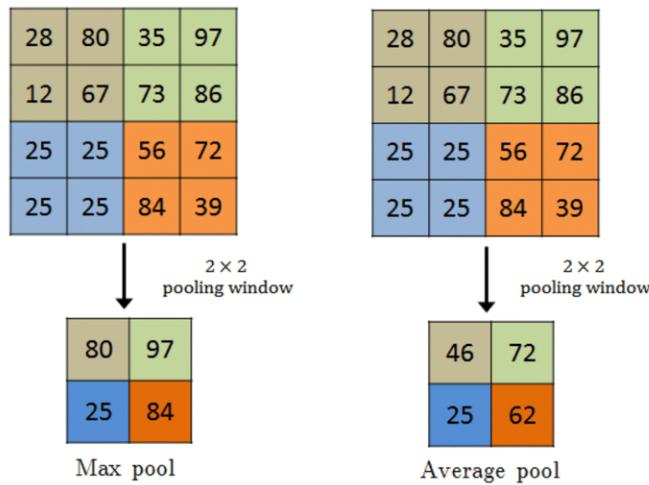


Figura 58: Operaciones de maxpooling y averagepooling⁴⁵

C.2.4 Capas upsampling y transposed convolution

Se utilizan principalmente en redes neuronales convolucionales cuyo objetivo es devolver una imagen en la salida. En una red neuronal convolucional típica, las capas de convolución y pooling reducen el tamaño de la entrada original de datos a medida que la información avanza a lo largo de la red. Sin embargo, en ciertas aplicaciones, como las redes de segmentación, es necesario que la salida mantenga las mismas dimensiones que la imagen de entrada para corresponder cada píxel con su clase asignada.

La operación de upsampling se encarga de re-escalar la matriz de entrada al tamaño deseado, calculando el valor de cada elemento mediante métodos de interpolación. Los métodos más utilizados para esta tarea incluyen:

- **Nearest Neighbor Interpolation:** Asigna a cada píxel re-escalado el valor del píxel más cercano en la imagen original (ver Figura 59).

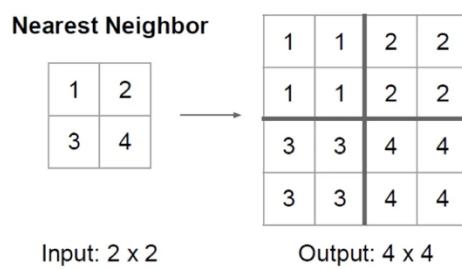


Figura 59: Técnica de Nearest Neighbor⁴⁶

⁴⁵ [https://www.researchgate.net/publication/340255272/figure/fig4/\[...\].png](https://www.researchgate.net/publication/340255272/figure/fig4/[...].png)

⁴⁶ https://miro.medium.com/v2/resize:fit:720/format:webp/1*9N9FVYalaVAk-aalcBVYaA.png

- **Bed of nails:** Se copian los valores de los píxeles de entrada y se colocan en la misma posición y se rellenan el resto de las posiciones de ceros (ver Figura 60).

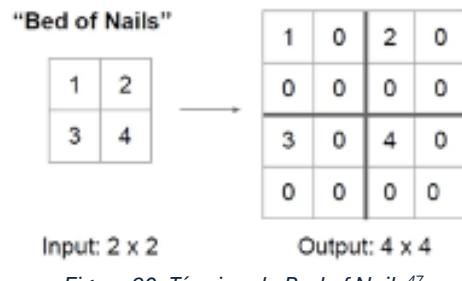


Figura 60: Técnica de Bed of Nails⁴⁷

- **Max-unpooling:** En esta operación, recordando las posiciones de los píxeles del max-pooling de una capa de pooling anterior, a la cual debe estar conectada, estos píxeles de la entrada se re-escalarán y se colocarán en las posiciones de los píxeles de la capa de max-pooling anterior a la que este conectada, y se rellena el resto de los píxeles de la matriz con ceros. Para comprenderlo mejor podemos observar la Figura 61.

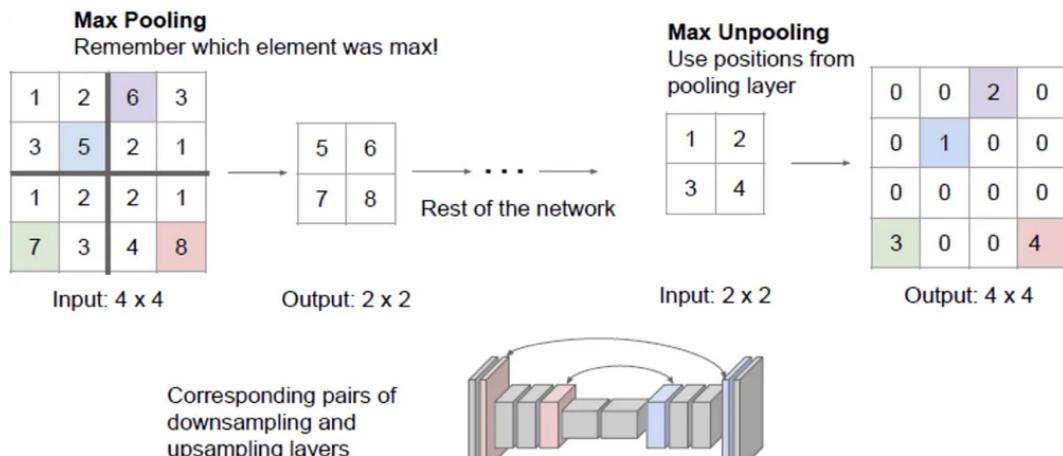


Figura 61: Técnica de maxpooling y maxunpooling⁴⁸

Sin embargo, en lugar de utilizar métodos de interpolación para aumentar los mapas de características, ya que nuestro interés radica en conservar las propiedades más importantes, podemos aplicar una operación que revierta los cambios efectuados por una capa de convolución. Esta operación es realizada por la capa de transposed convolution.

Para comprender mejor esta operación y los resultados que produce, primero plantearemos la operación de convolución de otra manera equivalente, mediante multiplicaciones de matrices.

Cuando realizamos una convolución sobre una imagen de tamaño 4x4 desplazando un kernel de tamaño 2x2, con un stride igual a 1 y sin padding, existirán 9 posiciones distintas en la imagen sobre las que el kernel se situará. Estas 9 ventanas se pueden representar como 9 matrices de 4x4 (el tamaño de la imagen) en las que todos los valores son 0, excepto en las posiciones donde

⁴⁷ https://miro.medium.com/v2/resize:fit:720/format:webp/1*LQVVqK8YJ4ndcU6NS4UOxA.png

⁴⁸ https://miro.medium.com/v2/resize:fit:1100/format:webp/1*b0NUJ-7IJnrljrzAc07BzQ.png

se sitúa el filtro de la convolución, que contendrán los valores del filtro. A continuación, estas 9 matrices se aplanan en vectores de una dimensión, y se concatenan creando una matriz de 9 filas y 16 columnas. Si aplanamos la imagen de entrada de 4x4 en un vector columna de longitud 16, podemos aplicar una multiplicación matricial entre la matriz descrita anteriormente y la imagen, resultando en un vector columna de longitud 9. La imagen resultante de la convolución, dados los parámetros mencionados y empleando la fórmula descrita, tendrá un tamaño de 3x3. Reorganizando el vector columna en una matriz de dimensión 3x3, obtenemos la imagen tras aplicar la convolución. Este proceso se ilustra en la Figura 62.

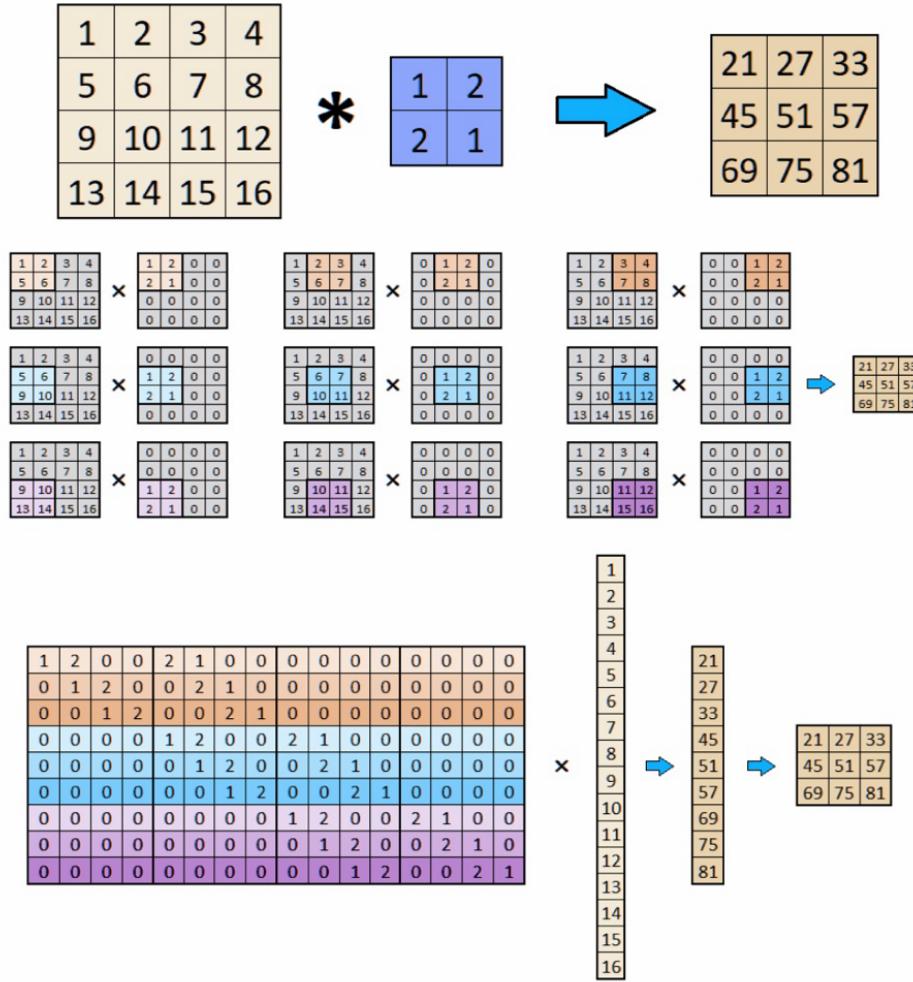


Figura 62: Operación de convolución como multiplicación de matrices

Una vez comprendido el proceso de la convolución como una simple multiplicación de matrices, podemos aplicar un enfoque similar para aumentar las dimensiones de un mapa de características, de modo que los nuevos valores dependan de un kernel cuyos valores se ajustan durante el entrenamiento de la red.

Si deseamos ampliar un mapa de características de tamaño 3x3 a 4x4, la matriz que realizará esta operación debe cumplir con un requisito fundamental: sus dimensiones. Como se describió anteriormente, las imágenes de entrada y salida se aplanan en forma de vectores de una dimensión. En este ejemplo, la entrada tendría una longitud de 9 y la salida una longitud de 16.

Por lo tanto, la matriz intermedia entre la entrada y la salida deberá tener 16 filas y 9 columnas, siguiendo la propiedad fundamental del producto de matrices. Existe una relación evidente entre las dimensiones de la matriz utilizada para realizar la convolución y la matriz utilizada para realizar la operación inversa. Ambas matrices tienen el mismo tamaño, pero sus dimensiones están intercambiadas, son transpuestas. Esta relación entre las matrices da nombre a la operación que estamos tratando: transposed convolution, explicada gráficamente en la Figura 63.

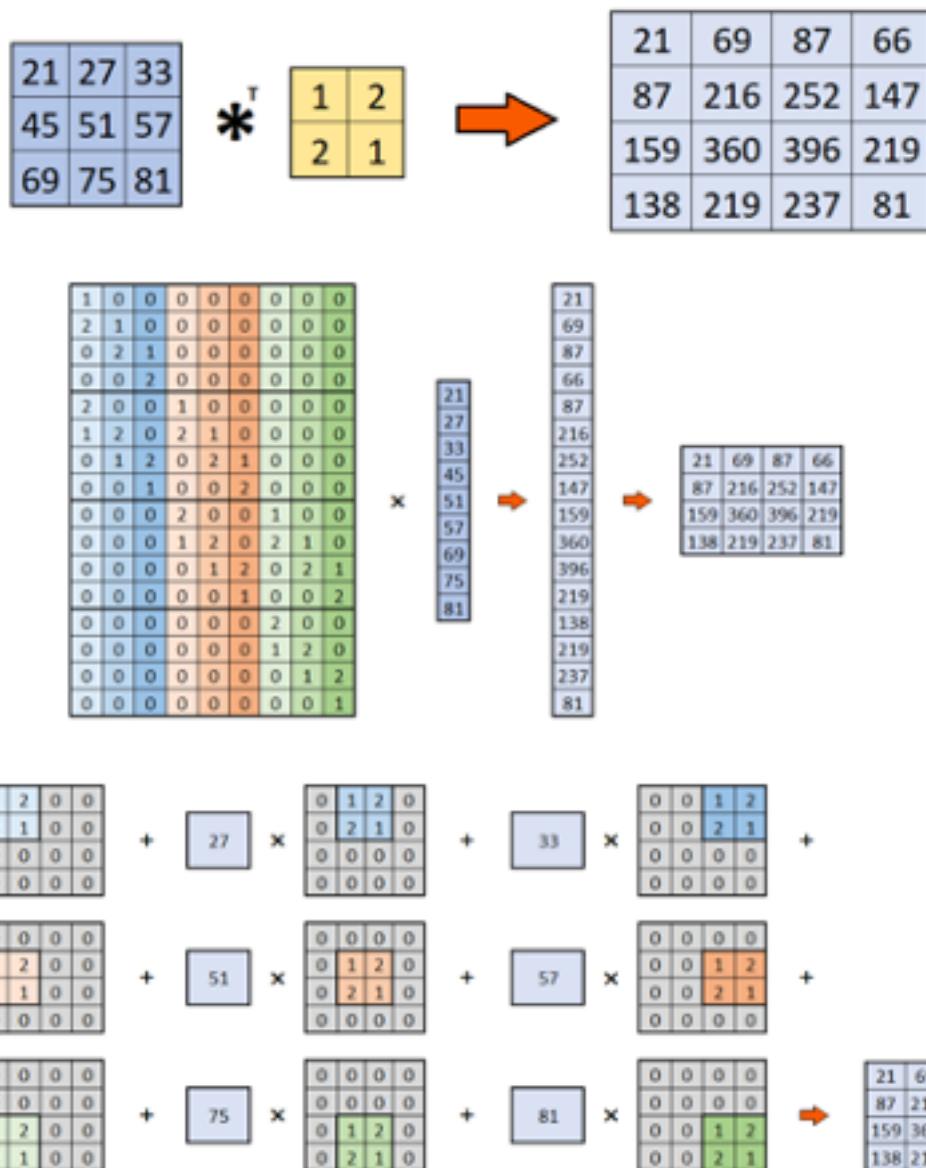


Figura 63: Operación de convolución traspuesta

C.2.5 Capas de softmax

Empleadas cuando se desea transformar la salida de una red neuronal en probabilidades, permitiendo así que la salida indique la probabilidad de que la entrada pertenezca a cada una de las clases. La operación que realiza la capa softmax se define como vimos en apartados anteriores, mediante la ecuación (4) vista anteriormente. Esta transformación asegura que las salidas sumen a uno y cada salida sea interpretada como una probabilidad, lo que hace que su uso resulte sencillo en tareas de clasificación en problemas con varias clases.

C.2.6 Capas fully connected

También conocidas como capas densas, son esenciales en los modelos diseñados para la clasificación de imágenes. Cada neurona de la capa está conectada con todas las neuronas de la capa anterior, suelen ubicarse al final de las redes neuronales convolucionales, después de que las capas de convolución y *pooling* hayan extraído y simplificado las características necesarias de la imagen de entrada.

El proceso generalmente implica que, antes de pasar de una capa convolucional a una capa *fully connected*, la salida convolucional (que es una matriz $M \times N$) pasa por una capa *flatten*. Esta capa *flatten* convierte la matriz en un vector unidimensional, preparándolo para la entrada en la capa *fully connected*.

Para configurar estas capas, ajustaremos los siguientes parámetros:

- **Número de neuronas:** Son las neuronas que forma la capa y definen el número de salidas de esta. El número de entradas de esta capa depende del tamaño que tengan las capas de convolución previas.
- **Función de activación:** Para esta función suele elegirse una de las que se usan en las capas de activación.

C.3 Arquitecturas populares

C.3.1 AlexNet

Como vimos en el Anexo teórico A, en su apartado A.2.2 Etapa Conexiónismo (1980 – 1995), AlexNet marcó un hito en la historia de la visión por ordenador al ganar la competición ImageNet, con una mejora significativa en precisión respecto a modelos anteriores.

Esta red introdujo una arquitectura más profunda y compleja que LeNet, compuesta por cinco capas convolucionales combinadas con capas de max-pooling y tres capas totalmente conectadas al final. Una de sus principales innovaciones fue el uso de la función de activación ReLU, que permitió una convergencia más rápida durante el entrenamiento. Además, implementó técnicas como dropout para reducir el sobreajuste y entrenamiento distribuido en

GPU, lo que facilitó el manejo de grandes volúmenes de datos. AlexNet supuso el inicio de una nueva era en el uso de redes neuronales profundas para la clasificación de imágenes a gran escala.

C.3.2 VGG

El modelo de aprendizaje profundo VGG fue desarrollado en 2014 por Karen Simonyan y Andrew Zisserman, investigadores del Visual Geometry Group de la Universidad de Oxford [19].

Su contribución más significativa radica en la demostración de que la implementación sistemática de convoluciones pequeñas (3×3) y capas con estructura repetitiva puede optimizar de manera notable el desempeño en tareas de clasificación de imágenes.

La red se presentó en dos versiones principales, VGG16 (ver Figura 64) y VGG19, con 16 y 19 capas de profundidad, respectivamente, y obtuvo el segundo lugar en la competición ImageNet de ese año. El modelo VGG emplea bloques de capas convolucionales seguidos de max-pooling, finalizando con tres capas fully connected y una capa Softmax para la clasificación. A pesar de que sus modelos poseen un elevado número de parámetros, VGG se distinguió por su simplicidad estructural y su eficacia, lo que la convirtió en una arquitectura base ampliamente utilizada en tareas de transfer learning y visión por computadora aplicada.

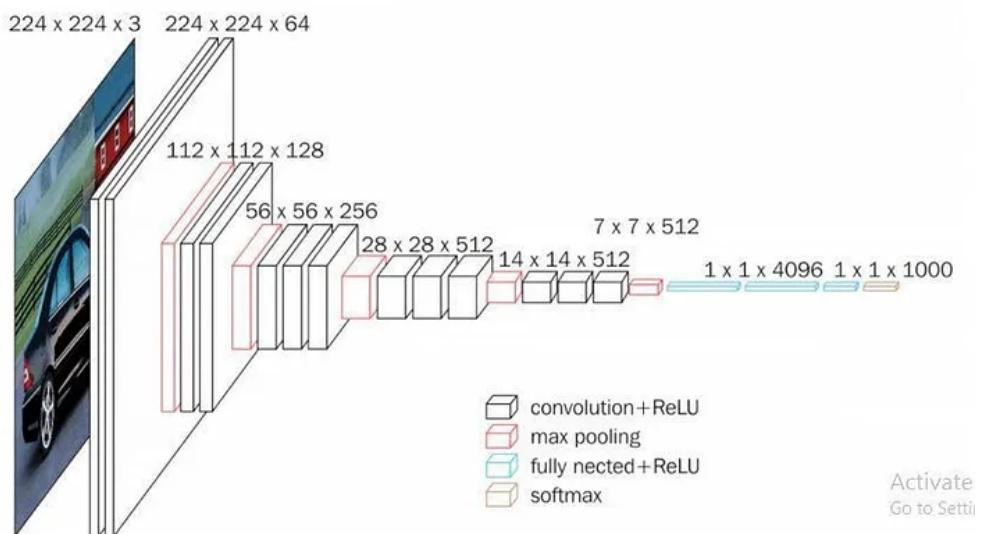


Figura 64: Arquitectura VGG16⁴⁹

C.3.3 GoogleNet

En el año 2014, el modelo de red neuronal GoogLeNet (ver Figura 65), también conocida como Inception V1, fue presentado por el grupo de investigación de Christian Szegedy [14]. Este modelo

⁴⁹ https://miro.medium.com/v2/resize:fit:720/format:webp/0*0M8CobXpNwFDCmOQ

resultó ganador del concurso ImageNet de ese mismo año, con un error de clasificación del 6,67 %.

La innovación principal consistió en la introducción del módulo Inception, que posibilita la realización de múltiples operaciones convolucionales de diversas dimensiones (1×1 , 3×3 y 5×5) de manera simultánea dentro del mismo bloque, combinando sus salidas de manera eficiente. Este avance posibilitó la concepción de una red profunda, pero computacionalmente optimizada, lo que resultó en una reducción significativa del número de parámetros en comparación con otras arquitecturas coetáneas. Además, se implementó el uso de global average-pooling en lugar de capas fully-connected, lo que contribuyó a disminuir el riesgo de sobreajuste y el consumo de memoria. Con una profundidad total de 22 capas, GoogLeNet representó un cambio de paradigma en el diseño de arquitecturas CNN, influenciando múltiples

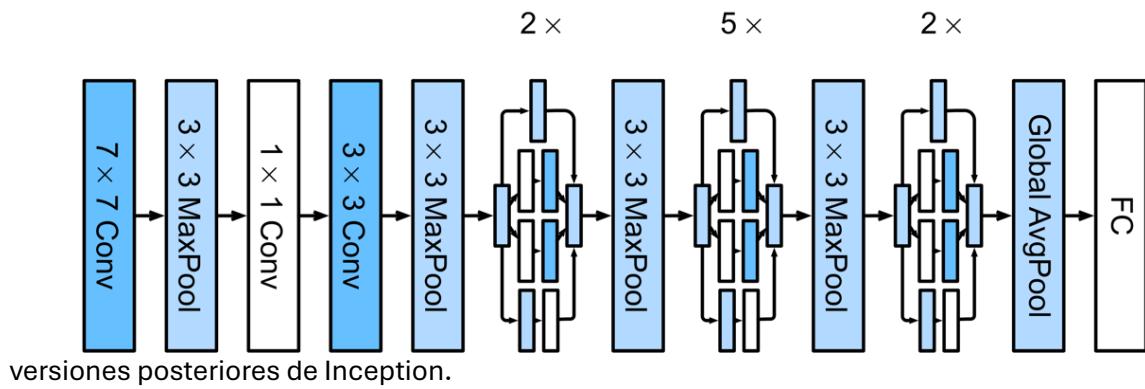


Figura 65: Arquitectura GoogleNet⁵⁰

C.3.4 ResNet

ResNet (Residual Network) fue introducida en 2015 por Kaiming He [15], y supuso un avance decisivo en el diseño de redes neuronales profundas al resolver eficazmente el problema del desvanecimiento del gradiente. Su arquitectura se basa en el concepto de conexiones residuales (skip connections), que permiten a la red aprender funciones residuales en lugar de funciones directas.

Estas conexiones permiten el flujo de información a través de la red y posibilitan el entrenamiento de modelos significativamente más complejos sin pérdida de rendimiento. La versión más conocida, ResNet-50, consta de 50 capas, aunque existen variantes más profundas como ResNet-101 y ResNet-152. ResNet-50 (ver Figura 66) obtuvo el primer puesto en la competición ImageNet 2015, alcanzando un error de clasificación del 3,57 %, y desde entonces se ha convertido en un modelo de referencia en múltiples aplicaciones de visión artificial, tanto en tareas de clasificación como de detección.

⁵⁰ [https://upload.wikimedia.org/wikipedia/commons/thumb/c/ca/\[%\]](https://upload.wikimedia.org/wikipedia/commons/thumb/c/ca/[%])

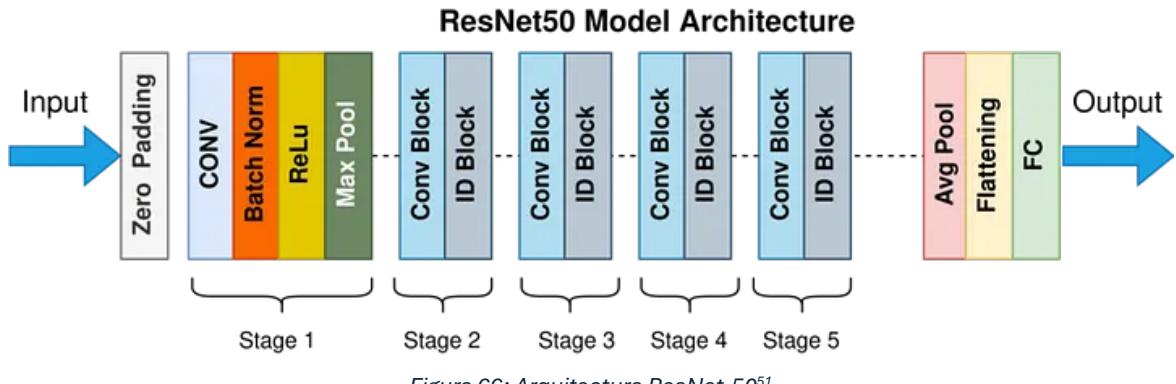


Figura 66: Arquitectura ResNet-50⁵¹

C.3.5 DenseNet

DenseNet (Red Neuronal Convolucional Conectada Densamente) fue presentada en 2017 por Gao Huang et al. como una evolución del diseño residual introducido en ResNet. Su característica principal es la conectividad densa entre capas, la cual se define como la capacidad de las redes neuronales de transmitir la salida de una capa a todas las capas siguientes dentro del mismo bloque.

Este enfoque ha demostrado una mejora significativa en la propagación del gradiente, fomentando la reutilización de características y reduciendo la redundancia de parámetros. Como resultado, se ha logrado entrenar modelos más eficientes en términos de memoria y computación. DenseNet ha demostrado un alto rendimiento en la clasificación de imágenes, logrando una excelente relación entre precisión y número de parámetros (ver). Se han propuesto variantes como DenseNet-121, DenseNet-169 y DenseNet-201, donde el número hace referencia a la profundidad de la red.

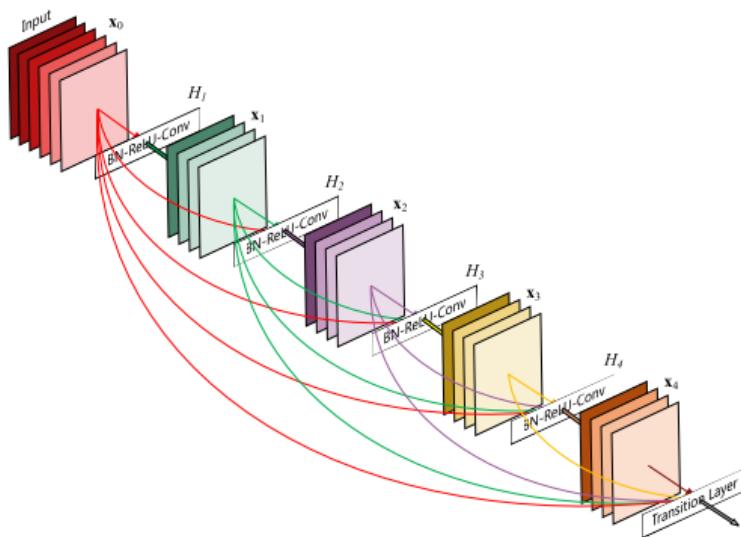


Figura 67: Arquitectura DenseNet⁵²

⁵¹ https://miro.medium.com/v2/resize:fit:720/format:webp/1*VM94wVftxP7wkiKo4BjfLA.png

⁵² <https://amaarora.github.io/images/densenet.png>

C.3.6 Arquitectura utilizada en este trabajo

Para el desarrollo de este proyecto, se implementó la arquitectura EfficientNet-B5, seleccionada tras una fase comparativa entre distintas redes preentrenadas disponibles en TensorFlow para aplicar técnicas de transfer learning con pesos ajustados sobre el dataset ImageNet. Esta evaluación incluyó arquitecturas como MobileNetV2, ResNet50, InceptionV3 y EfficientNet en sus diferentes variantes.

EfficientNet-B5 exhibió un equilibrio notable entre precisión, eficiencia computacional y capacidad de generalización, lo que la posicionó como la opción más apropiada para los objetivos planteados. Su escalado compuesto de profundidad, anchura y resolución permite maximizar el rendimiento sin incurrir en un coste computacional excesivo, lo que resulta especialmente ventajoso en contextos donde se requiere un alto nivel de precisión sin disponer de recursos de cómputo ilimitados.

Con el propósito de asistir a toda aquella persona que esté empleando este documento como referencia para su proyecto, se ha elaborado un resumen que detalla las principales arquitecturas que se consideran candidatos para el desarrollo del proyecto, junto con sus métricas más relevantes. Este resumen se ha elaborado con el objetivo de facilitar la toma de decisiones en función de las necesidades específicas de cada caso. Toda esta información puede verse en Tabla 15: Tabla comparativa de las principales arquitecturas disponibles en Tensorflow.

Retomando nuestra arquitectura seleccionada, EfficientNet-B5 es una de las variantes de la familia EfficientNet, introducida en 2019 por Mingxing Tan y Quoc V. Le [20]. Esta arquitectura se basa en el escalado compuesto, una técnica que ajusta de manera equilibrada la profundidad, anchura y resolución de la red para mejorar la eficiencia y precisión.

EfficientNet-B5 hace uso de bloques MBConv con convoluciones invertidas y módulos Squeeze-and-Excitation (SE), lo cual optimiza la extracción de características con un bajo costo computacional. Con una resolución de entrada de 456x456 píxeles, este modelo exhibe un alto rendimiento en ImageNet, destacándose por su equilibrio entre precisión y eficiencia en comparación con arquitecturas más extensas y onerosas en términos computacionales.

Tabla 15: Tabla comparativa de las principales arquitecturas disponibles en Tensorflow⁵³

Modelo	Precisión Top-1 (%)	Precisión Top-5 (%)	Tamaño de imagen	Nº de parámetros
MobileNetV2	71.8%	91.0%	224x224	3.5M
ResNet50	76.2%	92.9%	224x224	25.6M
InceptionV3	77.9%	93.7%	299x299	23.8M
VGG16	71.3%	89.8%	224x224	138M
VGG19	71.7%	90.0%	224x224	143.7M
DenseNet121	74.9%	92.2%	224x224	8M
EfficientNetB0	77.1%	93.3%	224x224	5.3M
EfficientNetB1	79.1%	94.4%	240x240	7.8M
EfficientNetB2	80.1%	94.9%	260x260	9.2M
EfficientNetB3	81.6%	95.7%	300x300	12M
EfficientNetB4	82.9%	96.4%	380x380	19M
EfficientNetB5	83.6%	96.7%	456x456	30M
EfficientNetB6	84.0%	96.8%	528x528	43M
EfficientNetB7	84.4%	97.0%	600x600	66M

La arquitectura EfficientNet se fundamenta en una serie de módulos que utilizan un enfoque optimizado para mejorar la eficiencia computacional y el rendimiento en la clasificación de imágenes. A continuación, se abordará la descripción de cada uno de los módulos que podemos ver en la Figura 68 y que forman la arquitectura de EfficientNet:

- **Módulo 1:** Se implementa una convolución separable en profundidad (Depthwise Conv2D), seguida de una normalización por lotes (Batch Normalization) y una función de activación. Posteriormente, se implementa una operación de promedio global-pooling, seguida de una capa de reescalado y dos convoluciones adicionales. Este módulo puede representar el bloque final de la red antes de la capa de clasificación.
- **Módulo 2:** Se aborda una temática similar a la del módulo anterior, si bien se incorpora el concepto de Zero Padding. Este último puede ser parte de un bloque convolucional con tamaños de filtro que requieren ajuste en los bordes de la imagen de entrada. Esta práctica es común en las etapas más profundas de la red.

⁵³ https://www.tensorflow.org/api_docs/python/tf/keras/applications

- **Módulo 3:** Un bloque característico de EfficientNet, donde se integran capas de convolución con normalización y reescalado. La implementación de Global Average Pooling sugiere que constituye un componente esencial en la etapa de clasificación final.
- **Módulo 4:** En este sentido, se evidencia la operación de multiplicación, lo cual es indicativo del mecanismo de Squeeze-and-Excitation (SE), utilizado en EfficientNet para mejorar la capacidad de la red en la selección de características relevantes. Se implementa una combinación de convoluciones con normalización por lotes para estabilizar el entrenamiento.
- **Módulo 5:** Este módulo presenta similitudes con el anterior, sin embargo, se ha incorporado Dropout, una técnica empleada para prevenir el sobreajuste. Esta observación sugiere que el módulo en cuestión pertenece a la porción final de la red, ubicada antes de la capa de salida.

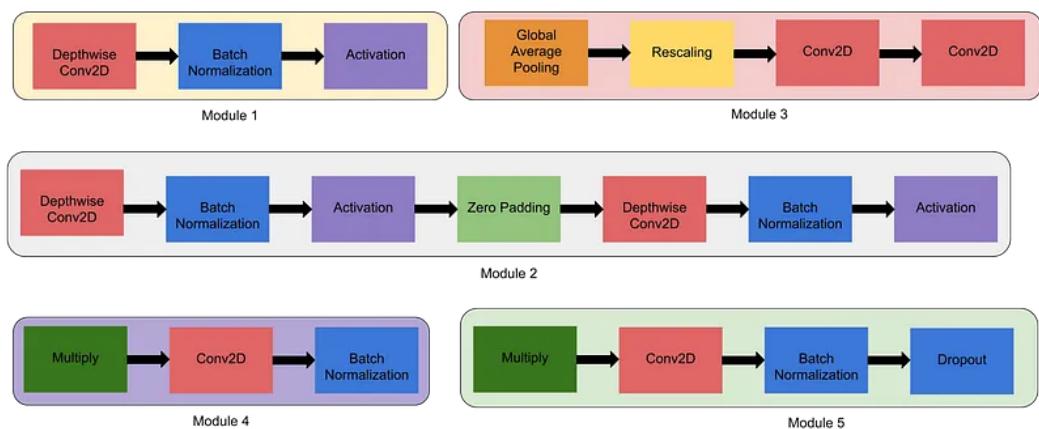


Figura 68: Tipos de modulos de EfficientNet⁵⁴

Una vez presentados los diversos módulos, es posible proceder a la comprensión de la configuración de la arquitectura EfficientNet-B5, así como de la organización de sus bloques de convolución y módulos internos, como se evidencia en la Figura 69.

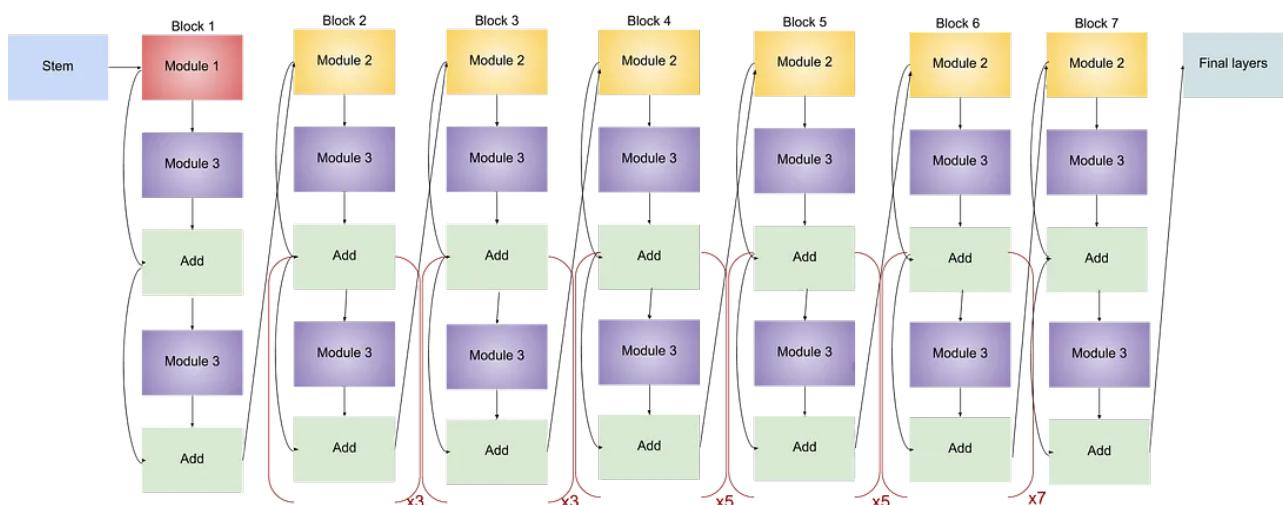


Figura 69: Arquitectura EfficientNet-B5⁵⁵

⁵⁴ https://miro.medium.com/v2/resize:fit:828/format:webp/1*cwMpOJNhwoeosjwW-usYvA.png

⁵⁵ https://miro.medium.com/v2/resize:fit:1100/format:webp/1*6vH0nsxj0_kHxF09tPFlg.png

Anexo teórico D MegaDetector

MegaDetector es un modelo de detección de objetos basado en aprendizaje profundo, desarrollado por Microsoft para facilitar el procesamiento automatizado de imágenes obtenidas mediante fototrampeo. Su objetivo principal es identificar la presencia de animales, personas y vehículos en estas imágenes, optimizando así el flujo de trabajo en proyectos de monitoreo de fauna silvestre a gran escala.

MegaDetector v5 está construido sobre la arquitectura YOLOv5 (You Only Look Once), una de las más eficientes y rápidas en tareas de detección de objetos. YOLOv5 es una red neuronal convolucional de tipo one-stage detector que procesa una imagen en una sola pasada, lo que permite realizar detecciones en tiempo real con una buena relación entre precisión y velocidad.

YOLO es una arquitectura de detección de objetos en tiempo real basada en redes neuronales convolucionales creada por Joseph Redmond como principal autor, su primera versión fue presentada en 2015 [21], a lo largo del tiempo Ultralytics ha ido publicando diferentes versiones y fue en 2020 cuando se lanzó la versión 5, convirtiéndose rápidamente en una de las versiones más populares de la familia YOLO por su velocidad, precisión y facilidad de uso.

La arquitectura de YOLOv5 incluye (ver Figura 70):

- **Backbone:** CSPDarknet, para extracción de características.
- **Neck:** PANet, para fusionar características a diferentes escalas.
- **Head:** capas de detección que generan cajas delimitadoras (bounding boxes) y clasificación por objeto detectado.

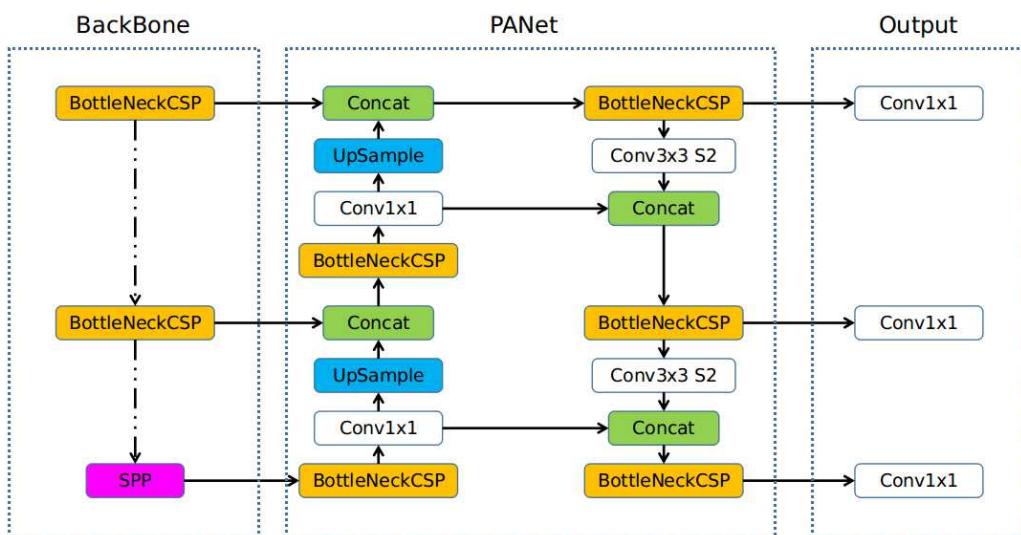


Figura 70: Arquitectura YOLOv5⁵⁶

⁵⁶ <https://user-images.githubusercontent.com/26456083/...>

MegaDetector adapta esta arquitectura con un entrenamiento específico sobre millones de imágenes etiquetadas provenientes de proyectos de conservación en todo el mundo, con especial énfasis en condiciones reales de campo (iluminación variable, occlusion, fondos naturales, etc.).

Está optimizado para detectar tres clases principales: Animales, humanos y vehículos. Cada detección incluye la posición en coordenadas de la caja delimitadora y una puntuación de confianza (confidence score) que representa la probabilidad asignada por el modelo a esa clase.

Como entrada utiliza un fichero de imagen, mientras que, como salida, genera un fichero JSON que contiene: Una lista de los objetos detectados, la clase a la que pertenece cada objeto, las coordenadas de cada detección en formato esquina superior izquierda, ancho y alto; y por último el valor de confianza de cada detección. Este formato permite integrar MegaDetector en flujos de trabajo personalizados, como filtrado automático de imágenes, visualización, anotación, y clasificación posterior.

MegaDetector v5 ha demostrado ser eficaz en una amplia gama de ecosistemas y configuraciones de cámaras trampa, con una alta tasa de detección en condiciones realistas. No obstante, su salida debe ser interpretada cuidadosamente:

- No distingue especies, solo detecta presencia.
- Puede generar falsos positivos o negativos, especialmente con animales parcialmente visibles o en condiciones de baja luz.
- El score de confianza puede ajustarse para equilibrar sensibilidad y precisión.

En este proyecto se ha utilizado MegaDetector como sistema de referencia para comparar su rendimiento con una CNN desarrollada para clasificar imágenes de fototrampeo en dos clases: presencia animal y ausencia animal. Las salidas del modelo se han interpretado como clasificación binaria, considerando la detección de al menos un objeto de cualquiera de las clases que maneja (animal, humano o vehículo) como indicativo de presencia, a raíz de estos valores se han podido generar el resto de métricas y poder realizar una comparación más detallada.