

Escuela Técnica Superior de Ingeniería Universidad de Huelva

Grado en Ingeniería Informática

Trabajo Fin de Grado

Aplicaciones de estrategias de Deep Learning para la
detección de animales en imágenes de foto-trampeo

Carlos García Silva

¿?/06/2025

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 comprensión, 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 foto-trampeo. 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*. La implementación de *MegaDetector* como preprocesamiento previo de las imágenes de entrada ha resultado en un rendimiento superior en comparación con otros modelos que no han hecho uso de dicha implementación.

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.

[TO-DO] [PÁRRAFO DE RESULTADOS].

[TO-DO] [PÁRRAFO FINAL DE CONCLUSIONES Y VALORACIÓN].

Abstract

The significance of a sustainable future is becoming increasingly evident within our society. Consequently, the study of biodiversity has assumed a pivotal role in the comprehension, preservation and estimation of life on Earth.

A prevalent technique in the execution of such research is the photo-trapping technique. This method involves the implementation of automated cameras within the designated study environment, with the objective of capturing images of animals within their natural habitat. A salient challenge associated with this approach pertains to the generation of voluminous image data, which renders manual analysis to be both costly and time-consuming.

The present Final Degree Project proposes a convolutional neural network with the capacity to classify photo-trapping images to identify those in which 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 implementation of MegaDetector as a pre-processing of the input images has resulted in a superior performance compared to other models that have not made use of such implementation.

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.

[TO-DO] [PÁRRAFO DE RESULTADOS].

[TO-DO] [PÁRRAFO FINAL DE CONCLUSIONES Y VALORACIÓN].

Índice

| | |
|--|-----------|
| Capítulo 1..... | 9 |
| 1.1 Motivación | 9 |
| 1.2 Objetivos..... | 10 |
| 1.3 Competencias..... | 11 |
| 1.4 Hardware y Software..... | 11 |
| 1.5 Organización de la memoria..... | 12 |
| Capítulo 2..... | 14 |
| 2.1 Historia y evolución del aprendizaje profundo | 14 |
| 2.1.1 Cibernética (1940 – 1960) | 15 |
| 2.1.2 Conexionismo (1980 – 1995) | 16 |
| 2.1.3 Deep Learning (2006 – Actualidad) | 18 |
| 2.2 Aprendizaje profundo en la actualidad..... | 19 |
| Capítulo 3..... | 23 |
| 3.1 Base de datos | 23 |
| 3.2 Conjuntos de datos | 23 |
| 3.3 Métricas de evaluación | 23 |
| Capítulo 4: Metodología | 24 |
| 4.1 Arquitecturas de la red | 24 |
| 4.2 Hiperparámetros | 24 |
| 4.3 Fase de entrenamiento | 24 |
| Capítulo 5: Resultados y Discusión..... | 25 |
| 5.1 Resultados en el conjunto de test | 25 |
| 5.2 Análisis y Discusión | 25 |
| Capítulo 6: Conclusiones y Trabajos Futuros | 26 |
| 6.1 Conclusiones técnicas..... | 26 |
| 6.2 Trabajos futuros | 26 |
| 6.3 Valoración personal | 26 |
| Referencias | 27 |
| Apartado 1: Introducción al Deep-Learning | 30 |
| Apartado 2: Redes Neuronales | 35 |
| 2.1 Fundamentos | 35 |
| 2.1.1 Perceptrón | 35 |
| 2.1.2 Funciones de activación | 37 |
| 2.1.3 Tipo de capas | 39 |
| 2.1.4 Aplicación a problemas de clasificación..... | 40 |
| 2.1.5 Aplicación a imágenes | 41 |
| 2.2 Desarrollo de redes neuronales | 42 |

| | |
|--|-----------|
| 2.2.1 Etapa de entrenamiento | 43 |
| 2.2.2 Etapa de entrenamiento: Hiperparámetros | 53 |
| 2.2.3 Etapa de entrenamiento: Control y seguimiento..... | 53 |
| 2.2.4 Etapa de inferencia: Aplicación y evaluación de la red | 53 |
| <i>Apartado 3: Redes Neuronales Convolucionales</i> | 54 |
| 3.1.- Introducción..... | 54 |
| 3.2.- Tipos de capas | 54 |
| 3.3.- Arquitecturas populares | 54 |
| 3.4.- Arquitectura utilizada en este trabajo | 54 |
| <i>Apartado 4: MegaDetector.....</i> | 55 |

Índice de figuras

| | |
|---|----|
| Figura 1: Principales frameworks de desarrollo de modelos de Deep Learning | 11 |
| Figura 2: Arquitectura de procesador M3 de Apple | 12 |
| Figura 3: Frecuencia de aparición de conceptos en publicaciones a lo largo de la historia | 15 |
| Figura 4: Demostración de cómo la función lógica XOR no puede modelarse linealmente | 16 |
| Figura 5: Representación del perceptrón multicapa y el algoritmo Backpropagation..... | 17 |
| Figura 6: Arquitectura LeNet | 18 |
| Figura 7: Arquitectura AlexNet | 19 |
| Figura 8: Rendimientos empresariales derivados de la IA por regiones (en millones de dolares) | 20 |
| Figura 9: Niveles de conducción autónoma | 20 |
| Figura 10: Etapas del proceso de Machine Learning..... | 31 |
| Figura 11: Esquema de aplicaciones del Machine Learning | 33 |
| Figura 12: Comparativa Machine Learning VS Deep Learning..... | 34 |
| Figura 13: Anatomía de una neurona biológica | 36 |
| Figura 14: Perceptrón y su proceso de aprendizaje..... | 36 |
| Figura 15: Estructura básica de una red neuronal multicapa | 39 |
| Figura 16: Transformación softmax y codificación one-hot | 40 |
| Figura 17: Ejemplo de flattening..... | 41 |
| Figura 18: Etapa de entrenamiento e inferencia de un modelo de machine learning | 42 |
| Figura 19: Ejemplo de matriz de confusión para 4 clases..... | 44 |
| Figura 20: Matriz de confusión considerando únicamente las clases positivas y negativas | 45 |
| Figura 21: Ejemplo de gráfica de Curva ROC..... | 47 |
| Figura 22: Representación gráfica de la dirección del gradiente..... | 50 |
| Figura 23: Comparación de algoritmos de optimización | 52 |

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.

El propósito de este estudio es examinar y desentrañar la implementación del *Deep Learning* en un amplio espectro de aplicaciones tecnológicas. En este sentido, se centrará en la visión artificial para concebir un sistema de detección de animales en imágenes de foto-trampeo. El enfoque metodológico seleccionado se fundamenta en la utilización de *MegaDetector*², aprovechando sus capacidades y descartando las áreas de escaso interés de las imágenes.

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

² Enlace oficial al modelo de detección: <https://github.com/microsoft/CameraTraps/tree/main>

Posteriormente, se implementa una red neuronal convolucional (CNN)³ para la clasificación de las imágenes, permitiendo distinguir entre aquellas que contienen animales y las que no. Este procedimiento se lleva a cabo con el propósito de mejorar la precisión y la eficiencia del proceso de monitoreo de la fauna en comparación con un sistema similar que no incorpora el preprocesamiento de las imágenes de entrada.

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 foto-trampeo 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 detección y clasificación eficaces de la presencia de animales en imágenes de foto-trampeo.

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*. 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.

³ Siglas del término inglés Convolutional Neural Network. Por motivos de simplicidad, durante el resto del trabajo utilizaremos dichas siglas en sustitución del término red neuronal convolucional.

⁴ Debido a su popularidad, a partir de ahora el término inglés *Dataset* sustituirá el término conjunto de datos / imágenes.

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. Específicamente, se ha concentrado en el estudio y la aplicación de las CNN para clasificación de imágenes de foto-trampero. 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 en profundidad 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.

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 *Machine Learning*⁵. (ver Figura 1).



Figura 1: Principales frameworks de desarrollo de modelos de Deep Learning

⁵ Término inglés cuyo significado se corresponde con el concepto de aprendizaje automático.

El marco de trabajo seleccionado para la ejecución de este proyecto es *TensorFlow*, 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 Keras, 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 Notebook y Visual Studio Code, junto con el entorno de desarrollo proporcionado por Anaconda, que nos permitirá instalar y gestionar fácilmente las librerías necesarias, como Tensorflow, OpenCV, Numpy y otras más.

Para la implementación del proyecto, se ha utilizado un MacBook Pro de 14 pulgadas, equipado con el chip Apple M3 (ver Figura 2)⁶, 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. La autonomía del dispositivo, con una duración de hasta 22 horas de batería, ha facilitado la realización de entrenamientos de larga duración sin interrupciones, optimizando el flujo de trabajo durante el desarrollo del proyecto.

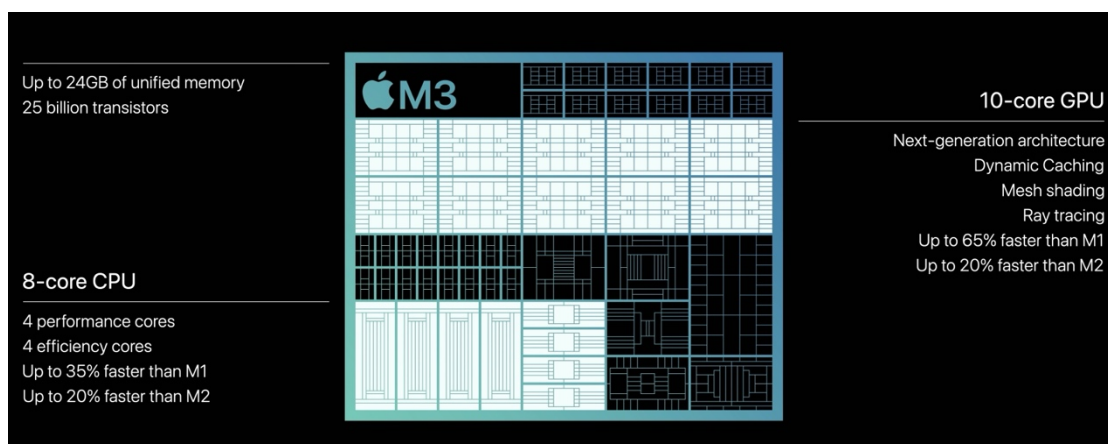


Figura 2: Arquitectura de procesador M3 de Apple

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:

⁶ Fuente: <https://www.notebookcheck.org/fileadmin/Notebooks/Apple/apple-m3-stats.jpg>

- **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.
- **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:** Ofrece una reflexión final sobre las implicaciones del estudio realizado y plantea posibles líneas de trabajo para futuras investigaciones. En este capítulo 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 el capítulo destinado a la bibliografía, en el cual se efectúan las referencias de los artículos, libros y recursos empleados para la elaboración del trabajo.

Además de los capítulos principales, la memoria contiene un anexo teórico adicional que profundiza en los conceptos fundamentales del trabajo. Organizado de la siguiente manera:

- **Apartado 1 Introducción al Deep Learning:** Se presentan los principios básicos del *Deep Learning*, destacando su evolución y relevancia en la actualidad.
- **Apartado 2 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.
- **Apartado 3 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.
- **Apartado 4 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

Una vez presentadas las motivaciones y objetivos, se procederá a proporcionar una visión integral sobre el *Deep Learning*, destacándolo como una disciplina especializada dentro de un campo más amplio, el *Machine Learning*. Para ello, se iniciará con una introducción a los conceptos fundamentales del aprendizaje profundo, seguida de un recorrido histórico, en el que se podrán conocer fundamentos teóricos y la evolución experimentada desde los años cincuenta hasta las avanzadas aplicaciones actuales.

Finalmente, se ofrece una revisión general de las numerosas aplicaciones del *Deep Learning*, demostrando su impacto significativo en diversos campos y su potencial para seguir transformando la tecnología y la sociedad.

2.1 Historia y evolución del aprendizaje profundo

El *Deep Learning* ha emergido como una de las áreas más innovadoras dentro del campo de la inteligencia artificial, ya que permite el desarrollo de sistemas capaces de resolver problemas de gran complejidad sin necesidad de definir explícitamente cada uno de los pasos a seguir. En el anexo teórico ([Apartado 1: Introducción al Deep-Learning](#)) se exponen 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 capítulo analizaremos la evolución histórica de las técnicas de aprendizaje automático, 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, el aprendizaje automático sirve de base para el aprendizaje profundo. 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.

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 3)⁷, 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.

⁷ Fuente: <https://books.google.com/ngrams/...>

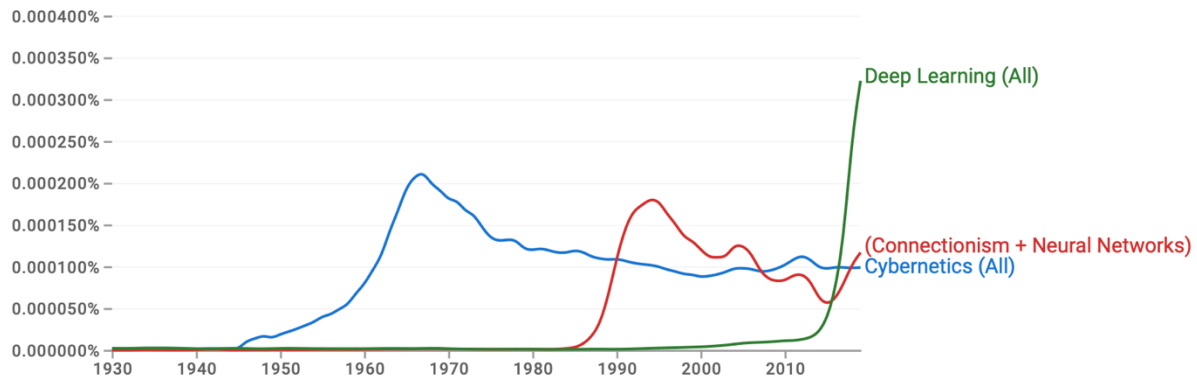


Figura 3: Frecuencia de aparición de conceptos en publicaciones a lo largo de la historia

2.1.1 Cibernética (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 AND⁸, OR⁹ y NOT¹⁰, 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.

Posteriormente, en 1958, Frank Rosenblatt propone el modelo del *perceptrón* (ver 2.1.1 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 ADELIN [5], que tenía la peculiaridad de permitir cuantificar el error y ajustar los pesos durante el entrenamiento en función del error cometido.

⁸ Operación lógica de disyunción.

⁹ Operación lógica de conjunción.

¹⁰ Operación lógica de negación.

Pese a todo ello, los modelos seguían siendo lineales. En 1969, Marvin Minsky y Seymour Papert [6] demostraron que los perceptrones no eran capaces de resolver funciones no lineales, como la función lógica XOR, también llamada OR exclusiva. (ver Figura 4)¹¹.

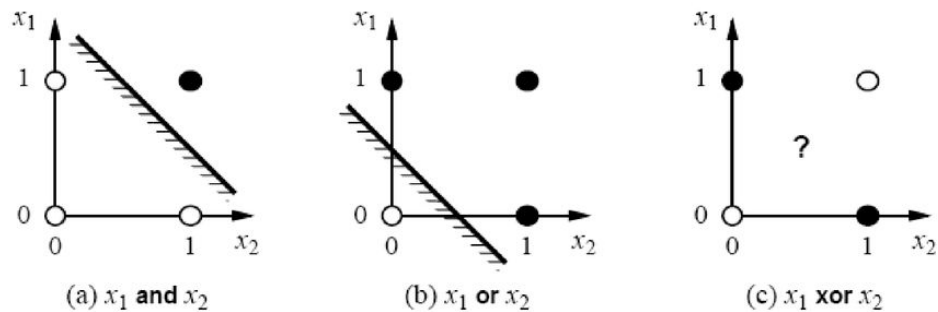


Figura 4: Demostración de cómo la función lógica XOR no puede modelarse linealmente

Esto ocasionó una gran decepción dentro de la comunidad y frenó el auge de esta tecnología, lo que conllevó un descenso del interés por su desarrollo en los años venideros.

2.1.2 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 5)¹².

¹¹ Fuente: <https://slideplayer.com.br/slide/14283270/89/images/47/...>

¹² Fuente: https://miro.medium.com/v2/resize:fit:1400/1*_ZpFxEmEkigpz3AzrmZYAQ.png

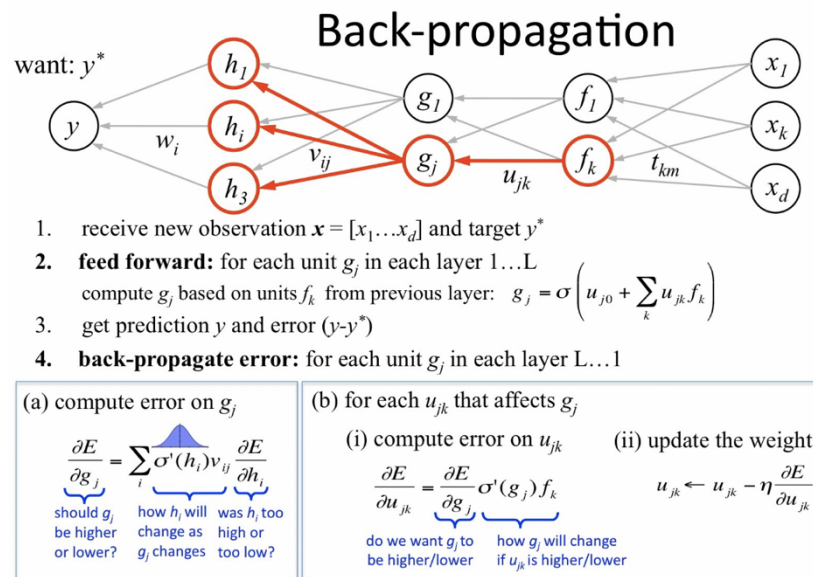


Figura 5: Representación del perceptrón multicapa y el algoritmo Backpropagation

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 (ver Figura 6 para mejor comprensión)¹³. 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 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.

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.

¹³ Fuente: https://miro.medium.com/v2/resize:fit:1400/1*bGjusyTjh2SACnkkMHU_hA.png

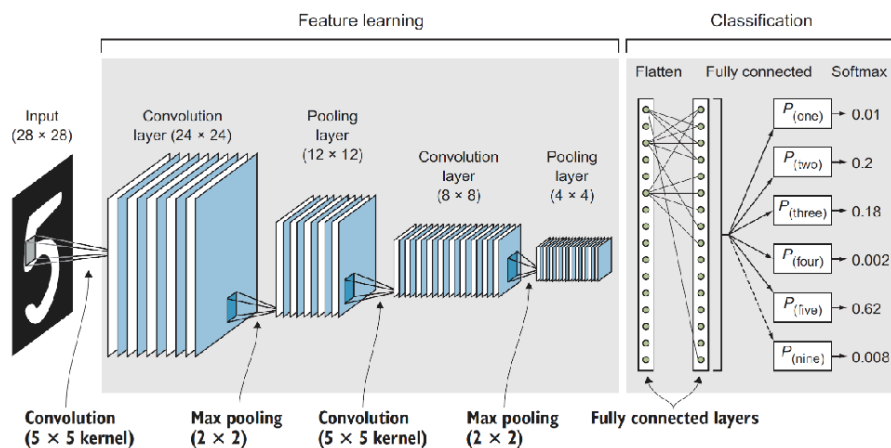


Figura 6: Arquitectura LeNet

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 máquinas vector soporte (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.

2.1.3 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 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 retropropagación 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* [13] (ver Figura 7)¹⁴ en la competición *ImageNet*. 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

¹⁴ Fuente: <https://www.researchgate.net/publication/320723863/figure/fig4/...>

clasificación de imágenes significativamente, bajándolo del 26 % a alrededor del 16 %. Este hito llamó la atención de muchos investigadores y grandes empresas.

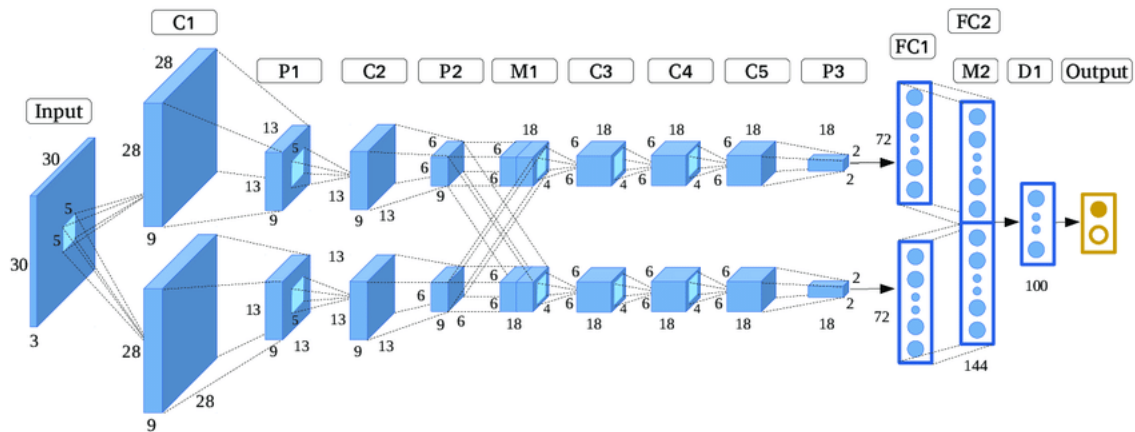


Figura 7: Arquitectura AlexNet

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 %.

2.2 Aprendizaje profundo en la actualidad

En la actualidad, el *Deep Learning* se ha convertido en una de las herramientas más utilizadas gracias a su capacidad para abordar una amplia variedad de problemas y su eficiencia.

Un factor clave para este hecho ha sido el desarrollo del hardware, en particular el de las tarjetas gráficas, que permiten procesar grandes cantidades de datos de manera eficiente y facilitan el entrenamiento de modelos complejos.

Una de las áreas donde ha tenido un impacto significativo es en la personalización del contenido y de las recomendaciones a los usuarios. Gracias al manejo de grandes cantidades de datos, lo que hoy en día se conoce como *Big Data*, muchas empresas han decidido diseñar modelos que se ajusten a las preferencias de sus usuarios y ofrecerles sugerencias personalizadas según el contenido que consumen.

El avance continuo de la potencia de cómputo y la reducción de costes ha hecho que el entrenamiento de modelos de *Deep Learning* sea cada vez más accesible, lo que impulsa aún más su integración en más sectores. En el ámbito del marketing, por ejemplo, el uso del Big Data para entrenar modelos que ajustan las ofertas y recomendaciones a los usuarios ha demostrado

ser extremadamente eficaz, lo que hace que servicios de *streaming* como *Netflix* sean un ejemplo del impacto positivo del *Deep Learning* en la experiencia de uso de los usuarios de estas plataformas. En la Figura 8¹⁵ se muestra el efecto del uso de la inteligencia artificial en los rendimientos empresariales.

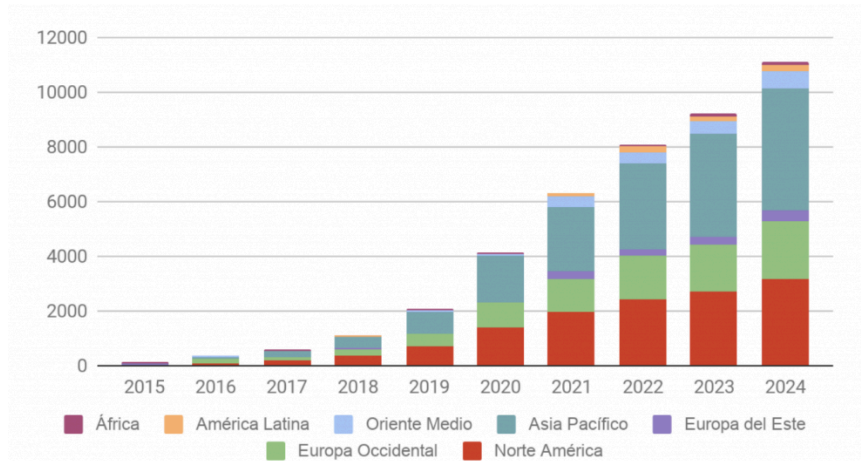


Figura 8: Rendimientos empresariales derivados de la IA por regiones (en millones de dólares)

Antes, todo en un videojuego era diseñado por equipos de personas: las mecánicas de los personajes, los entornos y los gráficos. Hoy en día, gracias a los avances en *Machine Learning*, la IA puede encargarse de muchas de estas tareas, lo que da lugar a mecánicas de juego más fluidas y entornos aleatorios. De este modo, los desarrolladores tienen mayor libertad, puesto que ahorran tiempo al no tener que diseñar los detalles manualmente. Además, la calidad de las imágenes ha mejorado, por ejemplo, NVIDIA ha desarrollado técnicas como DLSS 2.0 (*Deep Learning Super Sampling*) [16], que permiten a las tarjetas gráficas renderizar imágenes de menor resolución y luego escalarlas a alta resolución. Esto hace que la carga de trabajo de la tarjeta gráfica sea menor, aumenta la tasa de fotogramas por segundo y mantiene la nitidez en la imagen, por lo que el usuario tiene una mejor experiencia de juego sin sacrificar el rendimiento.

Otro sector en el que la inteligencia artificial ha cobrado importancia es el de la automoción, donde se ha producido una evolución significativa desde los sensores que detectan cuando un vehículo se sale de la vía hasta los coches que pueden estacionar solos. En la actualidad, hay una gran carrera entre las múltiples compañías automovilísticas para lograr el primer vehículo completamente autónomo, que eliminaría la necesidad de intervención humana. (ver Figura 9)¹⁶.

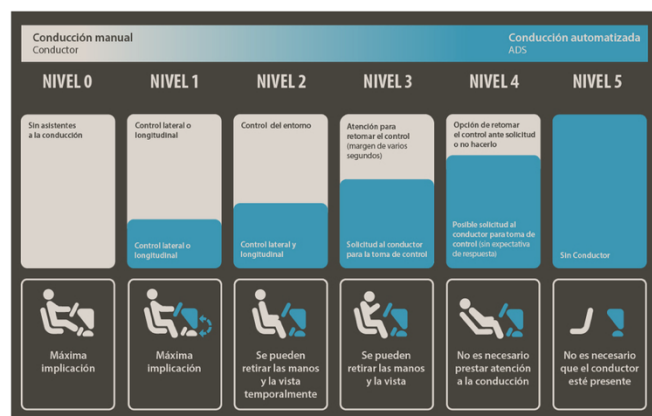


Figura 9: Niveles de conducción autónoma

¹⁵ Fuente: <https://ost.torrejuana.es/wp-content/uploads/2019/05/...>

¹⁶ Fuente: <https://www.km77.com/images/medium/5/7/9/5/2autonomo-km77-3-.335795.jpg>

Este sector está cada vez más cerca de alcanzar esta meta. Muchas marcas están desarrollando su propia inteligencia artificial con este propósito. Sin embargo, se trata de un tema delicado, ya que la seguridad y la vida de los ocupantes dependen de la precisión de la IA y no es posible prever ni entrenar todas las situaciones reales. De momento, solo se han implementado los primeros niveles, dependiendo del país y la legislación. De cara al futuro, aún se deben superar ciertos desafíos para que esto se convierta en una realidad común, algo que parece estar cada vez más cerca.

Los asistentes virtuales, como *Siri*, de Apple, *Alexa*, de Amazon, y *Google Assistant*, se han integrado profundamente en nuestra vida diaria gracias a los avances de la inteligencia artificial. Inicialmente, podían realizar tareas simples como poner una alarma o agregar una nota al calendario, pero han evolucionado enormemente y hoy en día pueden hacer llamadas para reservar establecimientos o controlar dispositivos de domótica en nuestros hogares. Su capacidad para entender y responder en lenguaje natural ha mejorado tanto que interactuar con ellos es casi como hablar con otra persona real. Esta evolución se debe a los constantes avances en el procesamiento del habla y a la recopilación continua de datos.

Además del avance en el procesamiento del lenguaje natural, la inteligencia artificial ha experimentado un crecimiento significativo en el reconocimiento y análisis de imágenes, lo que ha permitido a grandes empresas desarrollar soluciones innovadoras en múltiples sectores. Google ha implementado la IA en Google Photos y Google Lens, lo que permite identificar objetos, textos y rostros en imágenes con gran precisión. Amazon, con su servicio Rekognition, ha optimizado la identificación de productos, rostros y contenido en imágenes y vídeos, beneficiando tanto a empresas como a organismos de seguridad. Microsoft, a través de su iniciativa AI for Good Lab, ha desarrollado MegaDetector, un modelo especializado en la detección de fauna en imágenes de fototrampeo, ampliamente utilizado en proyectos de conservación, disponible en su repositorio Pytorch-Wildlife¹⁷ disponible en la plataforma GitHub [17].

Aprovechando estos avances, este trabajo propone una solución que combina la potencia de *MegaDetector* como preprocesador de imágenes con una red neuronal convolucional basada en la arquitectura *EfficientNet-B5*, optimizada mediante *Transfer Learning* con pesos preentrenados en *ImageNet*. *MegaDetector* ha demostrado su gran eficacia en la identificación de fauna en imágenes de foto-trampeo. Por otro lado, *EfficientNet-B5* ha recibido el reconocimiento generalizado de la comunidad científica por su capacidad para mejorar la precisión en la clasificación de imágenes y optimizar el uso de los recursos computacionales.

La integración de estos enfoques optimiza el proceso de clasificación de imágenes de fototrampeo, ya que mejora la precisión en la detección de animales y reduce la carga computacional del sistema. En este trabajo, *MegaDetector* se utilizará como un preprocesador, cuyo objetivo principal será identificar y descartar las zonas de la imagen que carecen de interés, de modo que la red *EfficientNet-B5* se centre únicamente en las regiones relevantes donde puede haber fauna. De esta manera, la CNN recibirá imágenes depuradas y optimizadas, lo que evitará el ruido innecesario en el entrenamiento y mejorará su capacidad para clasificar correctamente las imágenes como “con presencia animal” o “sin presencia animal/vacía”. A diferencia de otros enfoques, en los que la red neuronal debe analizar la imagen completa, este método facilita una mejor extracción de características, lo que optimiza tanto el rendimiento como la eficiencia del modelo.

¹⁷ Enlace del repositorio: <https://github.com/microsoft/CameraTraps/>

En estudios previos, se ha reconocido el preprocesamiento de imágenes para resaltar la región de interés como un paso fundamental en el rendimiento de las redes neuronales convolucionales. Sin embargo, muchos de los enfoques existentes realizan esta segmentación manualmente, como en el caso del trabajo de identificación de especies de roedores [18], lo que implica un proceso laborioso y poco escalable. Otros enfoques recurren a una CNN entrenada específicamente para la detección de regiones de interés, como en el estudio de identificación y conteo de animales salvajes [19], lo que implica una fase adicional de entrenamiento que incrementa la complejidad computacional.

A diferencia de estas aproximaciones, este trabajo propone la automatización del preprocesamiento mediante el uso de *MegaDetector*, un modelo previamente entrenado para la detección de animales en imágenes de foto-trampeo. Esto permite filtrar automáticamente las zonas irrelevantes de las imágenes y proporcionar únicamente la información relevante para el entrenamiento de la CNN a utilizar. Como resultado, se reduce significativamente la carga computacional y se optimiza el uso de recursos al evitar la necesidad de entrenar una CNN adicional para la detección. Además, la red clasificatoria se entrena mediante *transfer learning*, lo que acelera el proceso de convergencia y mejora la generalización del modelo con menos datos y en menos tiempo.

La ventaja clave del transfer learning consiste en que los modelos preentrenados ya han aprendido representaciones generales de las características visuales (bordes, texturas, formas) a partir de grandes volúmenes de datos, como *ImageNet*, lo que permite reutilizar ese conocimiento y adaptarlo a una tarea específica con un ajuste mínimo, lo que se conoce como *Fine Tuning*, argumento expuesto en muchos artículos de la comunidad [20]. Esto no solo reduce drásticamente el tiempo y los recursos computacionales necesarios para el entrenamiento, sino que también mejora la capacidad de generalización y evita problemas cuando se trabaja con conjuntos de datos más pequeños o especializados.

En conjunto, esta metodología supone una mejora respecto a los enfoques tradicionales, ya que automatiza el preprocesamiento, optimiza el flujo de entrenamiento y reduce los requerimientos computacionales sin comprometer la precisión de la clasificación. Esta innovación facilita la implementación de modelos de detección de fauna más escalables y eficientes, lo que permite su aplicación en estudios de biodiversidad de manera más accesible y reproducible.

Capítulo 3

Materiales

3.1 Base de datos

3.2 Conjuntos de datos

3.3 Métricas de evaluación

Capítulo 4: Metodología

4.1 Arquitecturas de la red

4.2 Hiperparámetros

4.3 Fase de entrenamiento

Capítulo 5: Resultados y Discusión

5.1 Resultados en el conjunto de prueba

5.2 Análisis y Discusión

Capítulo 6: Conclusiones y Trabajos Futuros

6.1 Conclusiones técnicas

6.2 Trabajos futuros

6.3 Valoración personal

Referencias

- [1 A. Mathison Turing, "On Computable Numbers, with an application to the
] entscheidungsproblem," 1936. [Online]. Available: <https://doi.org/10.1112/plms/s2-42.1.230>.
- [2 W. S. McCulloch and W. H. Pitts, "A logical calculus of the ideas immanent in nervous activity,"
] 1943. [Online]. Available: <https://doi.org/10.1007/BF02478259>.
- [3 A. M. Turin, "Computing Machinery and Intelligence," 1950. [Online]. Available:
] <https://doi.org/10.1093/mind/LIX.236.433>.
- [4 F. Rosenblatt, «The Perceptron: A Probabilistic Model for Information Storage and
] Organization in the Brain,» 1958. [En línea]. Available:
<https://psycnet.apa.org/doi/10.1037/h0042519>.
- [5 B. Widrow, "An Adaptive 'ADALINE' Neuron Using Chemical 'Memistors,'" 1960. [Online].
] Available: <https://isl.stanford.edu/~widrow/papers/t1960anadaptive.pdf>.
- [6 M. L. Minsky y S. A. Papert, «Perceptrons: An introduction to computational geometry,» 1969.
] [En línea]. Available: <https://direct.mit.edu/books/monograph/3132/PerceptronsAn-Introduction-to-Computational>.
- [7 D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Representations by Back-
] Propagating Errors," 1986. [Online]. Available: <https://doi.org/10.1038/323533a0>.
- [8 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).
- [9 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>.
- [1 S. Hochreiter, «Untersuchungen zu dynamischen neuronalen Netzen,» 1991. [En línea].
0] Available: <https://www.bioinf.jku.at/publications/older/3804.pdf>.
- [1 G. E. Hinton, S. Osindero and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," 2006.
1] [Online]. Available: <https://doi.org/10.1162/neco.2006.18.7.1527>.
- [1 G. E. Hinton, «Training Products of Experts by Minimizing Contrastive Divergence,» 2002. [En
2] línea]. Available: <https://doi.org/10.1162/089976602760128018>.
- [1 A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet Classification with Deep Convolutional
3] Neural Networks,» 2012. [En línea]. Available:
https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

- [1 C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke y A. 4] Rabinovich, «Going Deeper with Convolutions,» 2014. [En línea]. Available: <https://doi.org/10.1109/CVPR.2015.7298594>.
- [1 K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» 2015. [En 5] línea]. Available: <https://doi.org/10.1109/CVPR.2016.90>.
- [1 A. Burnes, «NVIDIA DLSS 2.0: Un gran salto en la renderización de IA,» NVIDIA, 23 Marzo 2020. 6] [En línea]. Available: <https://www.nvidia.com/es-es/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/>.
- [1 A. Hernandez, Z. Miao, L. Vargas, R. Dodhia y J. M. L. Ferres, «Pytorch-Wildlife: A Collaborative 7] Deep Learning Framework for Conservation,» 2024. [En línea]. Available: <https://arxiv.org/pdf/2405.12930>.
- [1 C. Seijas, G. Montilla y L. Frassato, «Identificación de especies de roedores usando 8] aprendizaje profundo,» 2019. [En línea]. Available: <https://doi.org/10.13053/cys-23-1-2906>.
- [1 G. Suing-Albito y L. R. B. Guamán, «Aplicación de métodos de deep learning en la 9] identificación y conteo de animales salvajes,» 2022. [En línea]. Available: https://www.researchgate.net/publication/371754966_Aplicacion_de_metodos_de_deep_learning_en_la_identificacion_y_conteo_de_animales_salvajes.
- [2 E. B. HENRÍQUEZ, «Transfer Learning en modelos profundos,» 2019. [En línea]. Available: 0] <https://telefonicatech.com/blog/transfer-learning-en-modelos-profundos>.
- [2 I. P. Borrero y M. E. G. Arias, «DEEP LEARNING Fundamentos , teoría y aplicación,» [En línea. 1] Available: <https://play.google.com/store/books/details?id=kzsvEAAAQBAJ>.
- [2 X. G. y. Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, 2] 2008.
- [2 K. He, X. Zhang, S. Ren y J. Sun, «Delving Deep into Rectifiers: Surpassing Human-Level 3] Performance on ImageNet Classification,» 2015. [En línea]. Available: <https://doi.org/10.1109/ICCV.2015.123>.
- [2 H. Robbins y S. Monro, «A Stochastic Approximation Method,» 1951. [En línea]. Available: 4] <https://doi.org/10.1214/aoms/1177729586>.
- [2 B. T. Polyak, «Some methods of speeding up the convergence of iteration methods,» 1964. [En 5] línea]. Available: [https://doi.org/10.1016/0041-5553\(64\)90137-5](https://doi.org/10.1016/0041-5553(64)90137-5).
- [2 D. P. Kingma y J. Ba, «Adam: A Method for Stochastic Optimization,» 2015. [En línea]. 6] Available: <https://doi.org/10.48550/arXiv.1412.6980>.
- [2 K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» 2015. [En 7] línea]. Available: <https://doi.org/10.1109/CVPR.2016.90>.

Apartado 1: Introducción al 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 extrapola 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.

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 10¹⁸, 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:

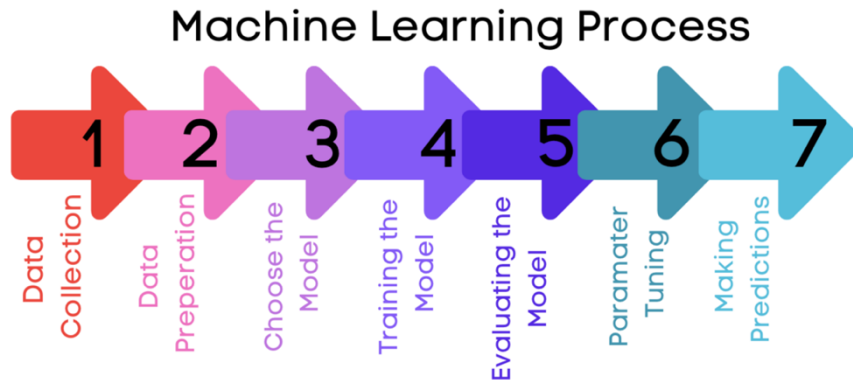


Figura 10: Etapas del proceso de Machine Learning

- **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 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

¹⁸ Fuente: <https://redmond.ai/wp-content/uploads/2023/05/word-image-583-2.png>

con diferentes *learning rate*¹⁹ para poder comparar cuál de ellos presenta un mejor comportamiento.

- **Etapas 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 SVM²⁰.
- **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 ID3²¹.
- **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 XGBoost²².

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 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.

¹⁹ Hiperparámetro clave en el entrenamiento de CNN que determina el tamaño de los ajustes en los pesos de la red.

²⁰ Support Vector Machine, algoritmo de clasificación que encuentra el hiperplano óptimo para separar distintas clases en un espacio de alta dimensión.

²¹ Iterative Dichotomiser 3, basado en árboles de decisión que construye el modelo seleccionando el atributo más informativo en cada división.

²² Extreme Gradient Boosting, algoritmo basado en árboles de decisión optimizados mediante boosting, diseñado para ser eficiente y preciso en tareas de clasificación y regresión.

- **Clasificación:** Su objetivo es modelar la relación 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 11²³.

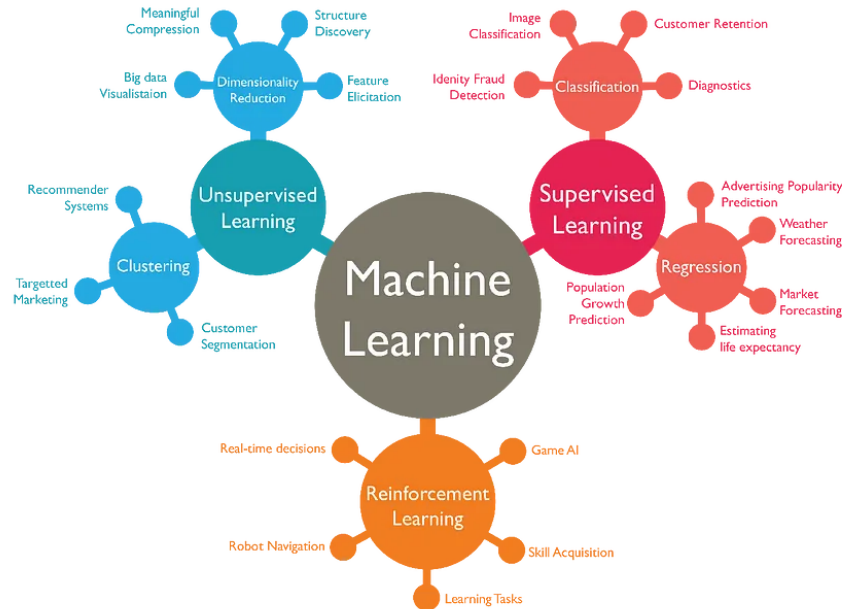


Figura 11: Esquema de aplicaciones del Machine Learning

- **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.
- **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.

²³ Fuente: https://media.licdn.com/dms/image/D5612AQHT5uzEz5mZ8g/article-cover_image-shrink_600_2000/0/...

- **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 10, 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 12)²⁴.

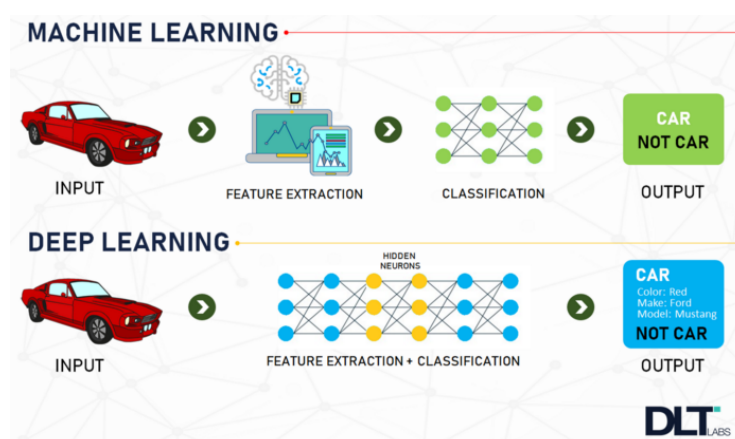


Figura 12: Comparativa Machine Learning VS Deep Learning

²⁴ Fuente: https://media.licdn.com/dms/image/D4D12AQH1_eSvKLwTiA/article-cover_image-shrink_600_2000/0/...

Apartado 2: 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 capítulo se presentan los fundamentos de las redes neuronales, abordando su evolución, estructura y funcionamiento.

2.1 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.

2.1.1 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 dendritas. 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 dendritas 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 13)²⁵.

²⁵ Fuente: <https://ml4a.github.io/images/neuron-anatomy.jpg>

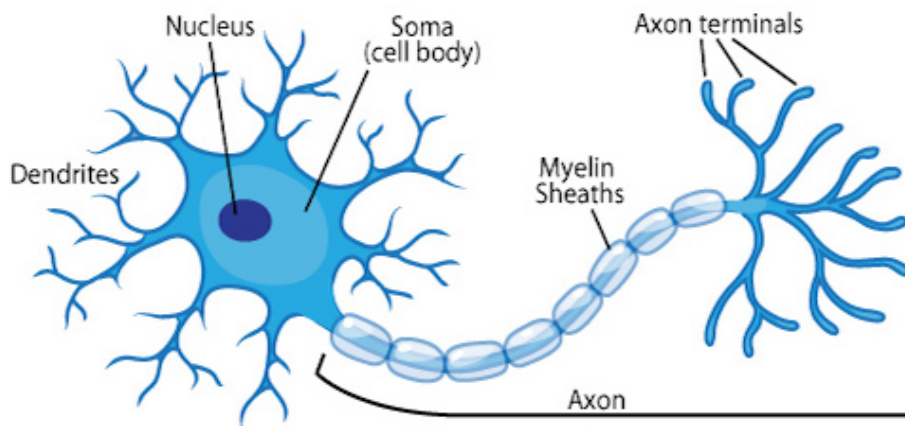


Figura 13: Anatomía de una neurona biológica

En este sentido, McCulloch y Pitts desarrollaron su modelo matemático, el cual recibe dos tipos de entradas binarias: excitadoras (x_i) e inhibitoras (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 inhibitora 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 inhibitoras. 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 14)²⁶.

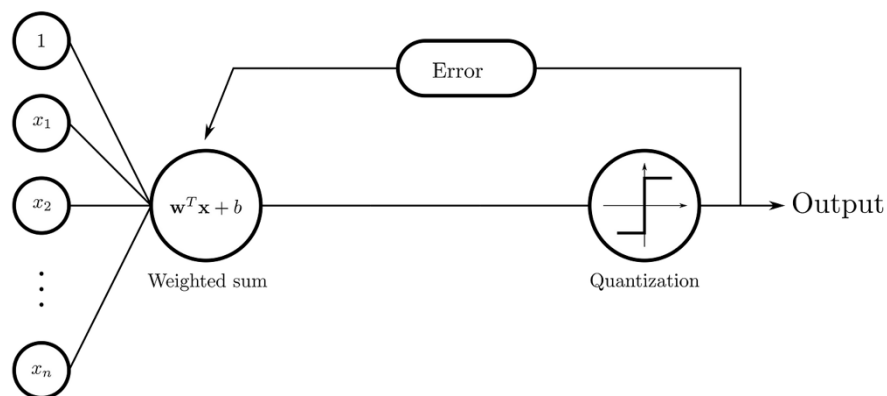


Figura 14: Perceptrón y su proceso de aprendizaje

²⁶ Fuente: https://miro.medium.com/v2/resize:fit:1400/1*BM8HuBXy9XYLvSGlBZwL0g.png

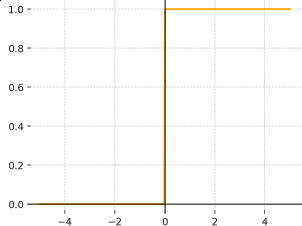
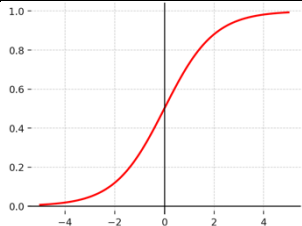
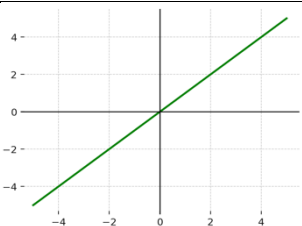
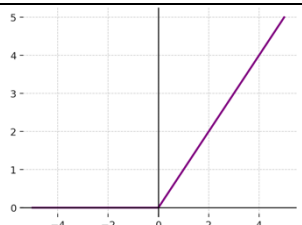
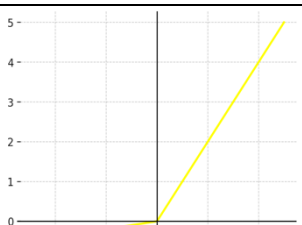
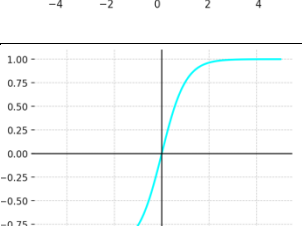
En 1960, Bernard Widrow y Marcian Hoff introdujeron el modelo ADALINE (Adaptive Linear Neuron) [5], que supuso una mejora significativa sobre el perceptrón al emplear una función de activación lineal en lugar de una función escalón. Este cambio permitió la aplicación de técnicas de optimización para 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.

2.1.2 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. (ver Tabla 1)

Tabla 1: 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 |  | $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ |

2.1.3 Tipo 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 15²⁷.

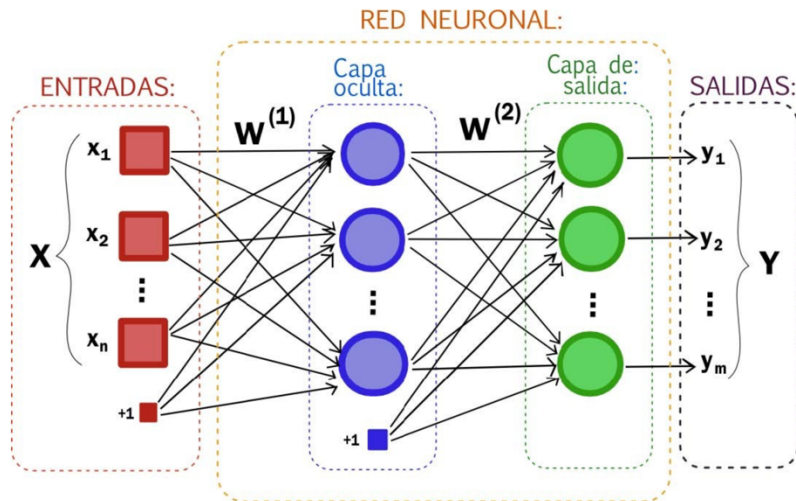


Figura 15: Estructura básica de una red neuronal multicapa

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

$$y = W^T x$$

Ecuación 1: Ecuación del perceptrón en formato vectorial

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 Ecuación 1, 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:

$$y = f(W^T x)$$

Ecuación 2: Ecuación del perceptrón aplicando función de activación a cada componente del vector

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

- **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

²⁷ Fuente: https://media.licdn.com/dms/image/D4E12AQHGrl3AHLPo2Q/article-cover_image-shrink_720_1280/0/...

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 2.1.1 Perceptrón, 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.

2.1.4 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.

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

Ecuación 3: Función softmax

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. (ver Figura 16) [21].

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

Figura 16: Transformación softmax y codificación one-hot

Al examinar la Figura 16, 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.

2.1.5 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 17)²⁸. 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 redes neuronales convolucionales (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.

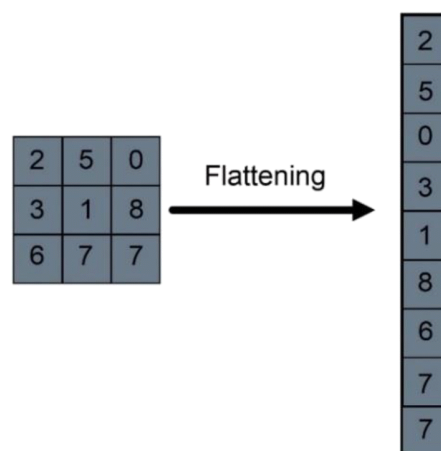


Figura 17: Ejemplo de flattening

²⁸ Fuente: <https://www.researchgate.net/publication/359174861/figure/fig5/...>

2.2 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 18²⁹.

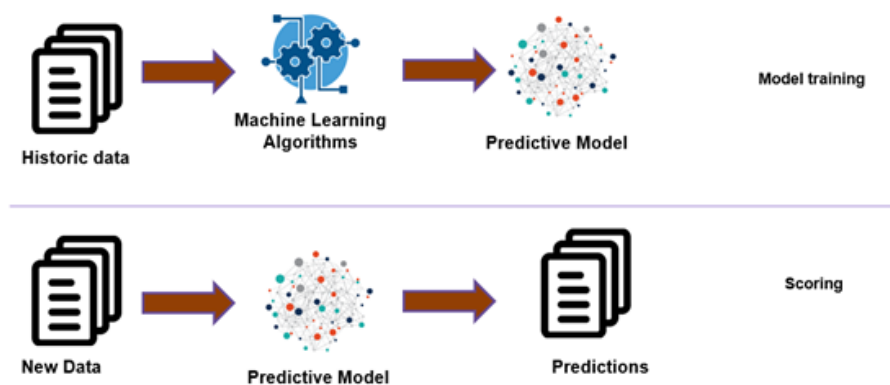


Figura 18: Etapa de entrenamiento e inferencia de un modelo de machine learning

- **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

²⁹ Fuente: <https://blogger.googleusercontent.com/img/b/R29vZ2xl/...>

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.

2.2.1 Etapa de entrenamiento

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.

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.

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 19)³⁰.

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

Figura 19: Ejemplo de matriz de confusión para 4 clases

³⁰ Fuente: <https://www.ibm.com/content/dam/connectedassets-adobe-cms/...>

En problemas de clasificación multiclase, 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 20)³¹.

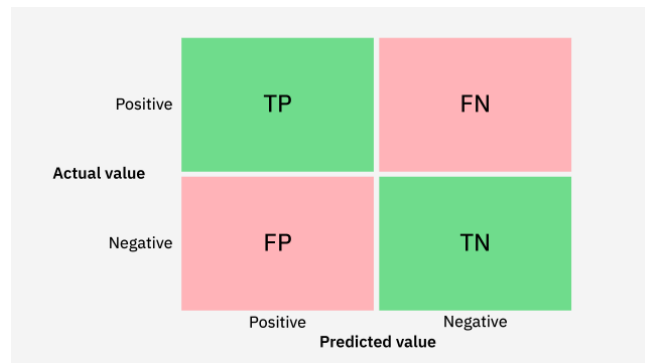


Figura 20: Matriz de confusión considerando únicamente las clases positivas y negativas

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:

- **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:

$$Precision = \frac{TP}{TP + FP}$$

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:

$$Recall = \frac{TP}{TP + FN}$$

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.

³¹ Fuente: <https://www.ibm.com/content/dam/connectedassets-adobe-cms/...>

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

$$Recall = \frac{TP}{TP + FN}$$

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:

$$F1\ Score = \frac{Precision \times Recall}{Precision + Recall}$$

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:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

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*).

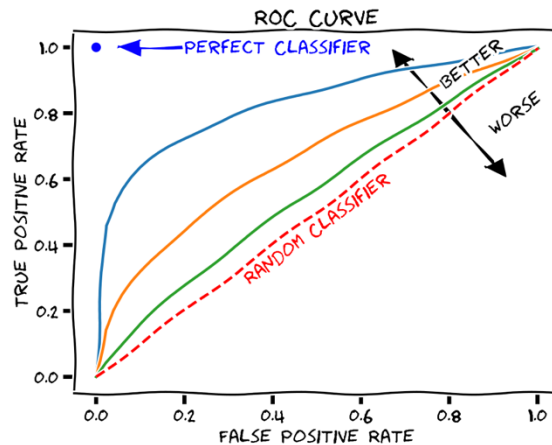


Figura 21: Ejemplo de gráfica de Curva ROC

Esta curva representa la tasa de verdaderos positivos (sensibilidad) en el eje Y, y la tasa de falsos positivos (especificidad) en el eje X. 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.

2.2.1.1 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:

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

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.

En problemas de clasificación multiclase, se utiliza la función conocida como *Categorical Crossentropy (CCE)*. Esta función calcula la probabilidad de pertenecer a cada una de las clases, se calcula de la siguiente forma:

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

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:

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

2.2.1.2 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 [21]. 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:

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

- **Inicialización de He:** Método creado en 2015 por los creadores de ResNet [22], 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:

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

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.

2.2.1.3 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:

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

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) (ver Figura 22)³². Este procedimiento da como resultado:

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

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.

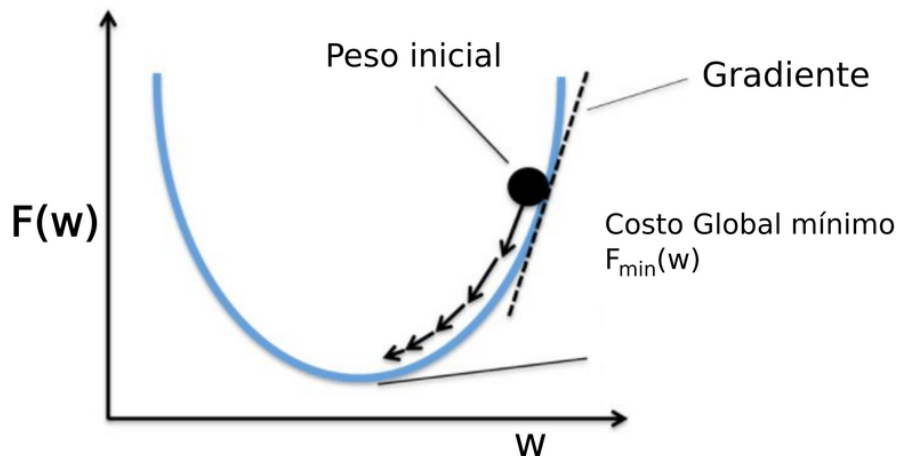


Figura 22: Representación gráfica de la dirección del gradiente

³² Fuente: <https://www.researchgate.net/profile/Juan-Vasquez-Gomez/publication/329453137/figure/fig10/...>

A esta técnica se le conoce como descenso por gradiente y es muy utilizada por los principales algoritmos optimizadores. (ver Figura 23)³³.

- **Stochastic Gradient Descent (SGD):** Introducido por Herbert y Sutton en 1951 [24], 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:

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

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 computo.

- **SGD with Momentum:** Con el paso del tiempo, Boris T. Polyak propuso su metodología, denominada Polyak's Heavy Ball Method [25], 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:

$$v_t = \beta v_{t-1} + (1 - \beta) \nabla_{\theta} J(\theta)$$

$$\theta_{nuevo} = \theta_{antiguo} - \alpha v_t$$

- **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.

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

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

³³ Fuente: https://tiddler.github.io/assets/img/2016-12-07-optimizers/cifar_cnn.png

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:

$$v_t = \rho v_{t-1} + (1 - \rho) [\nabla_{\theta} L(\theta)]^2$$

$$\theta_{nuevo} = \theta_{antiguo} - \frac{\alpha}{v_t + \epsilon} \nabla_{\theta} L(\theta)$$

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

- **Adam:** Adaptive Moment Estimation, es un algoritmo presentado en 2014 por Diederik Kingma [26], 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:

$$v_t = \beta_1 * v_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta w_t = -\alpha \frac{v_t}{\sqrt{s_t + \Delta_{\epsilon}}} * g_t$$

$$w_{t+1} = w_t + \Delta w_t$$

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

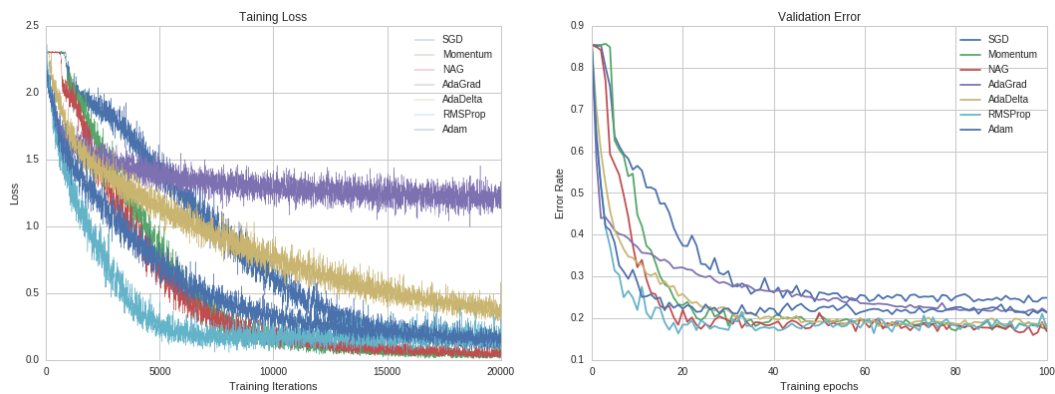


Figura 23: Comparación de algoritmos de optimización

2.2.1.5 Backpropagation

2.2.2 Etapa de entrenamiento: Hiperparámetros

2.2.3 Etapa de entrenamiento: Control y seguimiento

2.2.4 Etapa de inferencia: Aplicación y evaluación de la red

Apartado 3: Redes Neuronales Convolucionales

3.1.- 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.

3.2.- Tipos de capas

3.3.- Arquitecturas populares

3.4.- Arquitectura utilizada en este trabajo

Apartado 4: MegaDetector