Universidad Internacional de La Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Comparativa de redes convolucionales para clasificación de imágenes en visión artificial

Trabajo Fin de Máster

Presentado por: Bausá Cano, Alberto

Director: Pedraza Gomara, Luis

Ciudad: A Coruña
Fecha: 15 de septiembre, 2021

Resumen

El aprendizaje automático y las redes neuronales, con la visión artificial, han sido algunos de los principales campos de investigación de los últimos años dentro de la IA. Además, la incorporación de estas redes a tareas de visión artificial ha propiciado una creciente proliferación de alternativas en arquitecturas, datasets y librerías.

Por ello, se plantea una comparativa que analice las similitudes y diferencias entre tres redes convolucionales, dos de ellas preentrenadas, evaluando su desempeño sobre un mismo dataset, y recolectando métricas que permitan realizar una interpretación de los datos y lograr conclusiones relevantes que puedan ser reaprovechadas en proyectos similares.

Una vez finalizado el estudio, se han logrado algunas pistas interesantes acerca de la utilidad de aplicar *transfer learning* para abordar la solución de un problema de clasificación de imágenes, comprobando que se debe seleccionar cuidadosamente qué modelo elegir, hasta qué nivel conservar dicha arquitectura, y cuánto entrenar cada capa.

Palabras Clave: aprendizaje profundo, visión artificial, redes convolucionales, transferencia de aprendizaje, clasificación de imágenes

Abstract

Machine learning and neural networks, with computer vision, have been some of the main fields of research in recent years within IA. In addition, the incorporation of these networks to computer vision tasks has led to a growing proliferation of alternatives in architectures, datasets and libraries.

Therefore, a comparison is proposed to analyze similarities and differences between three convolutional networks, two of them pretrained, evaluating their performance on the same dataset, and collecting metrics that allow an interpretation of the data, and achieve relevant conclusions which can be reused in similar projects.

Once the study was completed, some interesting clues have been obtained regarding the usefulness of applying *transfer learning* to address the solution of an image classification problem, checking that it is necessary to be careful about making some choices, such as which model to choose, up to what level preserve said architecture, and how much train each layer.

Keywords: deep learning, computer visión, convolutional networks, transfer learning, image classification

Índice de contenidos

Índice de figuras

1. Introducción

En este capítulo se hará un breve repaso por la estructura del presente trabajo, esquematizando lo esencial de cada parte, lo cual permitirá adquirir una idea clara de lo pretendido con el mismo, así como las conclusiones que se han alcanzado, y el procedimiento seguido para ello.

Comenzaremos hablando de la problemática en la que se encuadra el estudio, justificando su relevancia actual, para continuar haciendo un esbozo de los objetivos perseguidos y la contribución que se espera realizar gracias a este. Con esa idea en mente, se dividirá el resto del capítulo en tres partes. La primera, para hablar de la motivación que ha impulsado el desarrollo de este trabajo, la segunda, para explicar el planteamiento y el proceso seguido para completar el estudio, y por último se describirán de manera breve el resto de capítulos de los que constará.

1.1 Motivación

Hoy día, la Visión Artificial es uno de los campos más potentes y con mayor proyección dentro del ámbito de la Inteligencia Artificial, con multitud de aplicaciones en el mundo real, y diversas técnicas y herramientas que permiten afrontar todo tipo de problemas: identificación y seguimiento de personas, clasificación de objetos en imágenes, detección de anomalías, reconocimiento de escenas, y en general, casi cualquier tipo de tratamiento o procesado que se le pueda realizar a una imagen.

Estrechamente enlazado a ese campo, tenemos otro igual de importante: el del Aprendizaje Automático, y en concreto, el Aprendizaje Profundo. Durante muchos años se han ido desarrollando y perfeccionando las herramientas utilizadas de forma mayoritaria en este último campo, las redes neuronales. Si bien es cierto que ha sido tan solo a lo largo de la última década que se ha empezado a popularizar el uso de un tipo específico de estas redes, conocido como red neuronal convolucional profunda, especializada en tareas propias del campo de la Visión Artificial.

Se pueden encontrar orígenes históricos de este tipo de redes convolucionales remontándonos a Fukushima (1979), quien hace más de 40 años desarrolló una red neuronal multicapa y jerarquizada, conocida como *Neocognitron*, compuesta por unidades que, pretendiendo imitar las células visuales del ojo humano, disponía de componentes especializados en reconocer patrones visuales en imágenes.

Años más tarde, el equipo de LeCun et al. (1989) conseguiría otros de los hitos de la disciplina, previo al auge que están teniendo todo este tipo de arquitecturas actualmente. Consiguieron aplicar con éxito un proceso de entrenamiento a la red, basado en retropropagación, y lo hicieron sobre la arquitectura propuesta anteriormente por Fukushima, construyendo así la primera CNN moderna, *LeNet-5*. Entre otros logros, este equipo también construyó uno de los primeros datasets con imágenes, *MNIST*.

Retomando la última década, se ha vivido un apogeo con estas arquitecturas profundas convolucionales, pudiendo encontrar infinidad de ellas, las cuales han ido apareciendo año tras año, aumentando de forma general su tamaño y complejidad. Una de sus principales bazas ha sido la de ir mejorando los resultados de sus predecesoras, y en muchos casos, introducir cambios en los modelos o técnicas aplicadas, que permitían ir progresando dentro del campo de manera veloz. Algunos ejemplos de estas numerosas arquitecturas serían *AlexNet*, *VGGNet*, *ResNet*, etc.

A su vez, se ha producido también una proliferación en cuanto a datasets disponibles para entrenar y evaluar estas arquitecturas, encontrándonos con *Fashion MNIST*, *CIFAR 10&100*, *ImageNet*… así como se han desarrollado multitud de herramientas que facilitaban todo el proceso, de principio a fin. Utilidades como el caso de *Tensorflow*, a día de hoy una de las plataformas o ecosistemas más utilizados en todo el mundo cuando se trata de implementar soluciones basadas en redes profundas.

Debido a todos estos factores, se logra comprender la magnitud y relevancia que tienen actualmente las redes neuronales convolucionales aplicadas a tareas de visión artificial. Entre esas tareas, destacan algunas, como la de la clasificación de imágenes. Tal extensión y diversidad de alternativas, hace que sea complicado seleccionar correctamente cuáles son las más adecuadas o se adaptan mejor a cada problema, haciendo necesaria la existencia de estudios, comparativas o análisis previos que permitan obtener cierto conocimiento o experiencia en el campo, con el fin de usarlo para realizar elecciones más conscientes e informadas.

1.2 Planteamiento del trabajo

Por tales motivos, partiendo de la problemática identificada y comentada anteriormente, se plantea el desarrollo de una comparativa entre algunas arquitecturas de estas redes convolucionales surgidas en los últimos años, restringiendo el alcance a tres de ellas, y un dataset objetivo sobre el que evaluar su rendimiento. Además, para ajustar todavía más el foco, la intención es focalizar el estudio en redes convolucionales especializadas en clasificar imágenes, tarea central de muchos datasets, como *CIFAR 10&100*, así como de algunas competiciones como la ILSVRC, con su dataset *ImageNet*.

El propósito principal es el de comparar arquitecturas entre sí, así como también diversas técnicas y formas de entrenarlas, distinguiendo dos aproximaciones: en primer lugar, una red implementada y entrenada completamente desde cero, segundo, la reutilización de un par de modelos diferentes de redes preentrenadas, adaptando cada uno al problema y dataset objetivo, y aplicando *transfer learning*.

Se comenzará haciendo una selección de los modelos, datasets y librerías más apropiadas para el estudio que nos ocupa, continuando con la búsqueda de las mejores implementaciones para esas arquitecturas seleccionadas. Sobre esa base, y después de construir las soluciones alternativas

propuestas, se ejecuta cada modelo en las distintas configuraciones de hiperparámetros previamente planteadas, con la finalidad de recolectar una serie de métricas bien definidas, entre las que podemos encontrar el valor de *accuracy* de la red, su pérdida (*loss*), así como el tiempo que tarda cada una para completar su entrenamiento, y qué impacto supone su consumo de memoria o espacio en disco.

Más tarde, estas métricas servirán como herramienta principal para la realización de un análisis acerca del rendimiento exhibido por cada modelo, y los resultados obtenidos, lo cual nos permitirá discutir las ventajas y desventajas de cada red, comparándolas entre sí y extrayendo conclusiones relevantes respecto a los comportamientos observados en cada caso. Este último paso será fundamental, pues de él saldrán muchas contribuciones con posibilidad de utilizarse en un futuro, las cuales permitirán inferir juicios acerca del desempeño de cada modelo.

1.3 Estructura de la memoria

Se enumerará aquí el listado de puntos de los que constará el estudio, con el objetivo de reseñarlos de manera concisa, para hacer un esbozo fiable de lo que se espera a lo largo de las siguientes páginas.

En primer lugar, y después de esta introducción, encontramos el **capítulo dos**, dedicado a describir el contexto científico y tecnológico en el que se enmarca todo el trabajo realizado, así como también a desglosar el estado del arte actual para la Visión Artificial, el Aprendizaje Profundo, y muy especialmente, las redes neuronales convolucionales, haciendo un amplio repaso de las principales arquitecturas surgidas en la última década.

A continuación, tendremos el **capítulo tres**, dedicado a describir el objetivo principal del trabajo, su motivación o razón de ser, desgranando dicho objetivo general en otros más específicos, que darán una idea de la metodología empleada y los pasos seguidos a lo largo del estudio.

Se prosigue con un **cuarto capítulo** dedicado al planteamiento de la comparativa que se pretende realizar, punto a partir del cual comienza el cuerpo principal que conforma la aportación o contribución del estudio. En él se empezará contando, de forma resumida, el proceso previo realizado para llegar desde la identificación del problema hasta la solución planteada, para después pasar a especificar dichas soluciones, explicando cada una de ellas y justificando su elección frente a otras alternativas. También se hará un repaso rápido por varias de las métricas empleadas.

Después del mencionado planteamiento, se abordará el desarrollo eficaz de la comparativa, a lo largo del **capítulo cinco**, en el cual se detallará cada una de las implementaciones, construidas mediante el lenguaje Python, para las diferentes arquitecturas. Asimismo, se presentarán los resultados como colofón, pero de manera aséptica, sin profundizar todavía sobre ellos.

Se finaliza esta aportación principal con el **capítulo seis**, en el cual se recoge una discusión y análisis minucioso de los resultados obtenidos anteriormente, repasando los valores de cada métrica, de cada red, y valiéndonos para ello de herramientas como tablas y gráficas, que permitan intuir de manera más sencilla algunas de las ideas que subyacen a los resultados obtenidos. Con las citadas ideas en mano, se razonan e interpretan posibles significados para esos valores, extrayendo conclusiones que puedan ser de utilidad en futuros proyectos de similar índole al aquí presentado. De esta forma, se consigue alcanzar un mejor entendimiento de las similitudes y diferencias, así como las ventajas y las desventajas, existentes entre los diferentes tipos de modelos propuestos.

Por último, tenemos un **capítulo siete** dedicado al cierre del trabajo, donde se sacarán algunas conclusiones acerca de la realización del mismo, haciendo un repaso que permita comprobar si se han cumplido, y en qué grado o con qué alcance, los objetivos planteados inicialmente. También se darán algunas ideas o vías para posibles trabajos futuros que puedan surgir a raíz de este.

2. Contexto y estado del arte

Este capítulo detalla y documenta el contexto científico en el que se desarrolla el presente trabajo, lo cual sirve como base conceptual e histórica para el resto de desarrollo presentado en el mismo.

Se comienza proporcionando algunas explicaciones o definiciones concisas de diversos conceptos estrechamente relacionados con el ámbito que nos ocupa, como son la Visión Artificial (*Computer Vision, CV*), el Aprendizaje Automático (*Machine Learning, ML*), y más específicamente el Aprendizaje Profundo (*Deep Learning, DL*). Además, se expone cómo, especialmente este último, tiene recientes aplicaciones en el terreno de la visión artificial, mediante nuevas técnicas y modelos como las Redes Neuronales Convolucionales (*Convolutional Neural Network, CNN*) o las Redes Generativas Antagónicas (*Generative Adversarial Network, GAN*), herramientas clave en su desarrollo reciente.

A partir de dichas definiciones, se hace un repaso histórico que permite encuadrar el marco teórico de referencia, mencionando algunas de las principales contribuciones en el área realizadas por diversos autores e instituciones, lo que sirve como trasfondo para contextualizar la problemática específica sobre la que versa el trabajo realizado. Para ello, también se mostrarán los desarrollos más recientes en el ámbito, exponiendo lo más novedoso dentro del estado actual de la técnica.

2.1. La visión artificial

Se puede considerar la visión artificial como aquella disciplina que combina el procesamiento digital de imagen con el reconocimiento de patrones, extrayendo como resultado alguna clase de comprensión o interpretación de la imagen analizada. En el caso de los algoritmos de visión artificial integrados con técnicas de aprendizaje automático, esa interpretación final de la imagen, cualitativa y cuantitativa, permite obtener información que guíe la toma de decisiones ante diversas situaciones.

Este análisis realizado a la imagen se puede dividir en varias fases (Wiley & Lucas, 2018):

- **Creación**. Se obtiene y almacena la imagen en un ordenador.
- **Preprocesamiento**. Se mejora la calidad de la imagen realzando los detalles de la misma que se quieran destacar, y se perfila el resultado aplicando técnicas como la eliminación de ruido.
- **Segmentación**. Se identifican los objetos significativos, separándolos de aquellos que no lo son, es decir, del fondo de la imagen, obviando todo lo que no sea de interés para el problema.
- **Extracción de métricas**. Se cuantifican características relevantes utilizando diversas medidas.

- **Interpretación**. Las métricas obtenidas son escrutadas para extraer conclusiones al respecto, como, por ejemplo, la categoría de un objeto reconocido en la imagen.

En la actualidad, algunas de las tareas o problemáticas más comunes, dentro del campo de CV, son las que se pueden apreciar en la Figura 1 (de izquierda a derecha).

- **El reconocimiento de objetos**, también conocido como clasificación de imágenes, donde dada una imagen, el sistema debe proporcionar como respuesta su categoría concreta, dentro de un listado acotado de posibles categorías conocidas.

- **La segmentación semántica**, o por conceptos. Esta tarea permite separar, mediante colores (por ejemplo), todos los píxeles de una imagen, etiquetando cada uno de ellos dentro de categorías predefinidas. En el caso de la figura, agrupa los píxeles hasta en 4 categorías principales: la hierba, el gato, los árboles y el cielo.

- **El reconocimiento (o identificación), sumado a la localización.** Si seguimos hablando de un único elemento, y añadimos la localización del objeto, tendríamos el tercer tipo de tarea, que permite no solo distinguir entre diferentes tipos de objetos y clasificarlos, sino también calcular y ubicar su posición en la imagen.

Por último, hablando de múltiples objetos en la imagen, podemos distinguir entre los siguientes casos:

- **La detección de objetos**. Permite extender la tarea anterior de clasificación y localización a múltiples elementos, permitiendo resaltar en una misma imagen, tanto la categoría de varios objetos detectados simultáneamente, como también su posición.

- **La segmentación de instancias**. Derivando el concepto de segmentación semántica, pero aplicado esta vez a objetos pertenecientes a diferentes categorías, tendríamos esta técnica que permite clasificar los píxeles de interés de una imagen, dentro de categorías de interés predefinidas, obviando el resto de los píxeles que no pertenezcan a una de esas categorías.

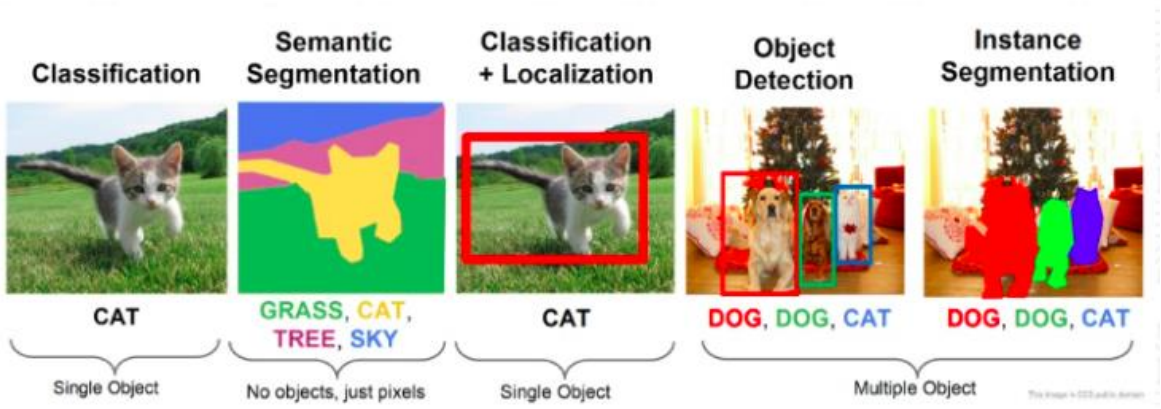

Figura 1. Tareas comunes en visión artificial.
Fuente: Fei-Fei Li & Justin Johnson & Serena Yeung (2017)

2.1.1. Evolución histórica

Uno de los primeros hitos, en cuanto a la historia y las aportaciones realizadas a la disciplina, sería sin duda el realizado por Hubel & Wiesel (1959), quienes publicaron uno de los artículos científicos más influyentes dentro del campo de la percepción visual. En él, describían propiedades fundamentales acerca de la respuesta neuronal del córtex visual, así como la forma en que la experiencia visual adquirida por los gatos daba forma a la arquitectura del mencionado córtex visual de los mismos.

Sus experimentos se basaban en la colocación de electrodos en el córtex visual primario de gatos anestesiados, y en la observación de la actividad neuronal del gato mientras le mostraban diferentes imágenes. Tras varios experimentos, llegaron a algunas conclusiones interesantes, como que dentro del córtex existían neuronas tanto simples (encargadas de respuestas a la orientación de la luz) como complejas (centradas en el movimiento y las formas). Además, comprobaron que el procesamiento visual siempre empezaba con estructuras simples, como por ejemplo la detección de aristas orientadas en cierta dirección.

En la misma época cabe destacar un par de aportaciones. Por un lado, está el trabajo de Kirsch et al. (1957), quienes desarrollaron una máquina capaz de transformar una imagen analógica, en una matriz de números que la describía. Se trataba del primer “escáner digital”, y permitió obtener la primera imagen digital de la historia: una fotografía de 5x5 cm del hijo de Kirsch, capturada como una matriz cuadrada de 176x176 píxeles.

Por otro lado, tenemos la máquina de percepción de sólidos tridimensionales, de Roberts (1963), considerada uno de los precursores de la visión artificial moderna. En su tesis, describía el proceso de derivar información tridimensional de objetos sólidos, a partir de fotografías en dos dimensiones, reduciendo el espacio a formas geométricas simples.

En la década de 1960 nació el *Summer Vision Project*, de la mano de Papert (1966), quien tuvo la ambiciosa idea de querer resolver el problema de la visión artificial, acompañado por un puñado de estudiantes del MIT, en un solo verano. Ese pequeño grupo logró construir un sistema automático capaz de realizar segmentación del fondo de una imagen respecto al objeto de interés (*ROI)*, y también consiguió extraer de la imagen objetos no solapados, a partir de fotografías del mundo real. Aunque el proyecto no fue un éxito, pues ciertamente no alcanzó el ambicioso objetivo inicial, es considerado como el nacimiento oficial de la disciplina de visión artificial como campo científico.

Casi dos décadas después, Marr (1982) realizó otra interesante e influyente contribución, basándose en las ideas aportadas por Hubel & Wiesel, de que el procesamiento visual comenzaba con formas o estructuras simples. Gracias a ello, Marr llegó a una importante conclusión: la visión era jerárquica, y su función principal era, partiendo de formas simples, crear complejas representaciones tridimensionales

del entorno para poder interactuar con él. Incluso llegó a construir un marco para la representación visual, con 3 niveles:

- Esbozo inicial. Se representan esas estructuras simples (bordes, límites, etc.).
- Esbozo $2^{1/2}$D. Aparecen superficies, información de profundidad, discontinuidades…
- Modelo 3D. Organizado jerárquicamente en términos de elementos fundamentales o primitivos de superficies o volúmenes.

Durante la década de 1990, el campo de la visión artificial sufrió un importante cambio de enfoque, mediante el cual los investigadores pasaron de intentar reconstruir objetos usando representaciones 3D de los mismos, siguiendo la senda propuesta por Marr unos años antes, a dirigir sus esfuerzos hacia el reconocimiento de objetos basado en características.

Uno de los trabajos más representativos de esa nueva corriente fue el de Lowe (1999), quien describía un sistema de reconocimiento visual que usaba características locales de los objetos, invariables a la rotación, traslación y cambios leves de iluminación. Para Lowe, estas características eran similares a las propiedades de las neuronas de la corteza temporal, encargadas de los procesos de detección de objetos propios de la visión primaria.

Poco después, Viola & Jones (2001) construyeron el primer sistema de detección de rostros que funcionaba a tiempo real. Se trataba de un clasificador binario *fuerte*, utilizando varios clasificadores *débiles*, entrenados usando Adaboost (técnica de *boosting*, Freund & Schapire, 1999). Seleccionaba un pequeño número de características visuales críticas, produciendo clasificadores extremadamente rápidos, que combinaba en cascada, permitiendo descartar áreas del fondo de la imagen, centrándose rápidamente en las regiones más prometedoras en relación al objeto de interés.

Una de las últimas contribuciones al campo fue el trabajo realizado por Felzenszwalb et al. (2008), el conocido como *Deformable Part Model* (DPM). Funcionaba descomponiendo objetos en una colección de partes constituyentes, y estableciendo un conjunto de relaciones geométricas entre ellas. Demostró gran rendimiento en el reconocimiento de objetos, superando otras técnicas como el *template matching*.

2.1.2. Acerca de los *datasets*

Ya en el nuevo milenio, y a medida que el campo de la visión artificial iba avanzando, la comunidad científica sentía la necesidad de disponer de métricas estándar de evaluación y *datasets* para pruebas de rendimiento, que permitiesen comparar los diferentes modelos que iban apareciendo y creándose en paralelo por diferentes grupos de investigación. De esta forma, fueron apareciendo paulatinamente algunos proyectos que permitieron hacerlo posible.

En el año 2005 nacía el *Pascal VOC project*[1], proporcionando un *dataset* estandarizado para clasificación de objetos en imágenes, así como un conjunto de herramientas para visualizar y manipular dicho *dataset* y su contenido. Contaba con un total de 9963 imágenes, cada una con un conjunto de objetos de hasta 20 categorías diferentes, haciendo un total de 24640 objetos anotados.

Se puede apreciar en la Figura 2 un ejemplo de clasificación de imágenes utilizando el mencionado dataset, donde se muestran un par de imágenes identificadas por cada categoría de las consideradas.

Figura 2. Ejemplos de imágenes clasificadas en Pascal VOC.
Fuente: San Biagio (2014)

Desde entonces, y hasta el año 2012, se organizó una competición anual que permitía evaluar y comparar el rendimiento obtenido por diversos métodos para el reconocimiento de clases de objetos.

Se trataría de la precursora de competiciones que surgirían a raíz de esta, como la *ImageNet Large Scale Visual Recognition Competition* (ILSVRC)[2]. En la Figura 3 se muestra una imagen sobre la cual se ha realizado la tarea de detección de objetos, dentro de este último certamen.

[1] Página web relativa al proyecto *Pascal VOC*. Debido a que la página oficial no está disponible, se ofrece esta web que sirve como respaldo del proyecto y del *dataset*: https://pjreddie.com/projects/pascal-voc-dataset-mirror/

[2] Página web oficial del ILSVRC, donde se evalúan algoritmos para la clasificación de imágenes y la detección de objetos a gran escala: http://image-net.org/challenges/LSVRC/

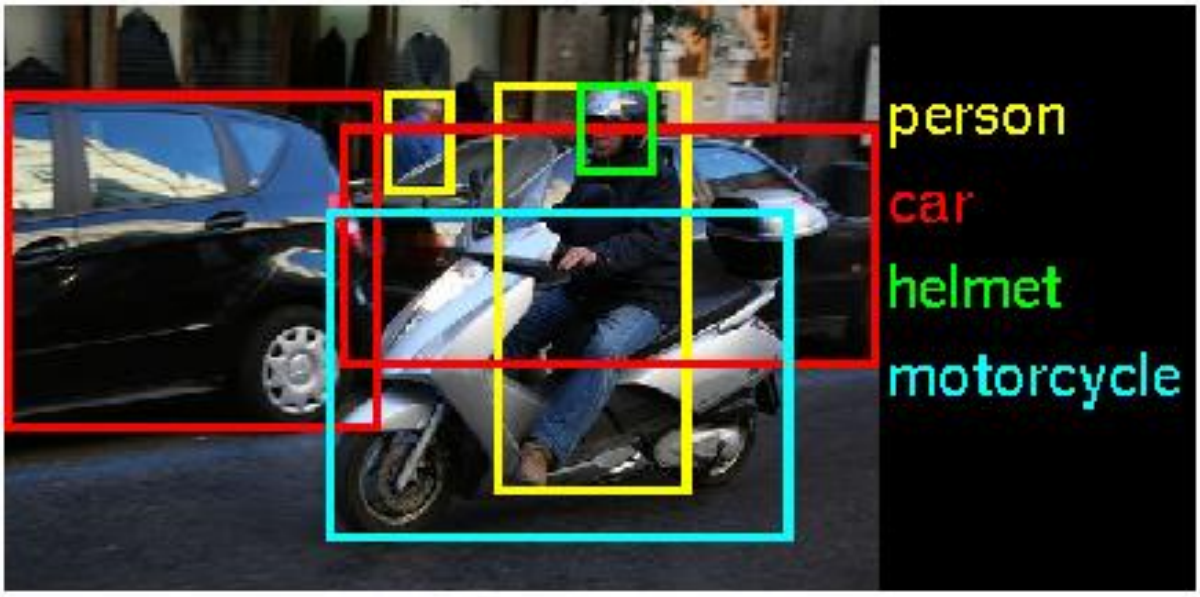

Figura 3. Ejemplo de detección de objetos realizado en el ILSVRC.
Fuente: Russakovsky et al. (2014)

Sobre la misma fotografía, se puede ver cómo se identifican los objetos, asignándoles una clase y un color, y además de ello, se define su localización exacta en la imagen mediante recuadros que indican su ubicación, tanto para diferentes objetos de diferentes clases, como para diferentes instancias de la misma categoría de objeto. Incluso, como se puede distinguir, el proceso realizado permitía diferenciar también objetos intercalados unos con otros, no mostrándose de forma completa y aislada, sino solapados, lo cual aumentaba la dificultad del análisis.

Así, en el ejemplo de la figura mostrado, se puede apreciar cómo el sistema es capaz de detectar (identificar y localizar) elementos como personas, coches, cascos de moto, y motocicletas, diferenciando claramente sus respectivas ROIs.

A pesar de los grandes avances realizados en el ámbito, aún estamos lejos de “resolver” el campo de investigación de la visión artificial, como pretendía Papert hace más de 50 años, pues como hemos visto, son múltiples los problemas a los que se enfrenta. Sin embargo, ya existen multitud de aplicaciones prácticas desarrolladas con los conocimientos actuales y empleadas en problemas del mundo real. Más recientemente, con la explosión de las redes neuronales, y más concretamente, de las CNNs y las GANs, se han producido avances si cabe más significativos, quizá impensables hace medio siglo, pero que hoy en día nos parecen ya cotidianos.

2.2. Aprendizaje automático

Se entiende por aprendizaje automático el subcampo, dentro de la Inteligencia Artificial y de las Ciencias de la Computación en general, que se ocupa de investigar y desarrollar técnicas que permitan que las máquinas aprendan. Es decir, que mejoren el comportamiento o rendimiento que exhiben, gracias a experiencia o conocimiento acumulado, adquirido mediante una fase de entrenamiento previa.

2.2.1. Evolución histórica

Para hablar de sus orígenes, podemos encontrar una de las primeras referencias a “máquinas que aprenden” en los trabajos de Turing (1950), donde exponía una característica fundamental de las mismas, y es que el *profesor* que enseñase a dichas máquinas, tendría una cierta ignorancia de qué podría estar pasando internamente, haciendo un símil con un modelo de caja negra, aunque sí que sería capaz de predecir parcialmente su comportamiento. Esto contrastaba en gran medida, decía, con las tareas de computación clásicas, para las cuáles se debía tener una idea clara del estado de la máquina en cada momento, y conocer los posibles caminos de ejecución.

Por otro lado, uno de los primeros trabajos categorizados específicamente como ML, fue el realizado por Samuel (1959), quien desarrolló un sistema capaz de jugar a las damas, proporcionándole al mismo únicamente las reglas del juego, así como alguna otra información útil que el propio sistema tenía que sopesar, y dejando que aprendiese por sí solo durante un periodo determinado de tiempo. Demostró, de esta forma, que existían principios de ML que podían ser aplicables de forma general, abriendo una nueva vía de investigación.

Una década después de Turing, Minsky (1961) trataba con problemas de búsqueda, probabilidad y aprendizaje, hablando abiertamente sobre ML y “sistemas que aprendían”. Mencionaba que, por sí mismos, estos sistemas solo podían ser útiles en situaciones recurrentes, con hechos conocidos, automatizando soluciones triviales. Sin embargo, si se combinaban con técnicas de clasificación o reconocimiento de patrones, el aprendizaje máquina permitía predecir situaciones completamente nuevas, ofreciendo una solución no trivial a problemas con situaciones desconocidas.

A partir de entonces, se vivió una época de menores descubrimientos para el ML y la IA en general, conocida como *invierno de la IA*. Se trató de 2 periodos caracterizados por una reducción de fondos para la investigación, unido a un desinterés general en el tema. Debido a esas épocas complicadas, durante los años 70 y 80 se produjo una separación entre una investigación en IA, más interesada en soluciones basadas en conocimiento (sistemas expertos), y una industria de ML enfocada a cuestiones eminentemente prácticas, usando conceptos de teoría de probabilidad y estadística.

Cabe destacar, como resultado de esas últimas décadas, el surgimiento de técnicas como los *boosting algorithms*, a raíz de trabajos como el de Schapire (1990). Estas nuevas herramientas permitían, mediante el uso de métodos propios de la probabilidad y la estadística, convertir sistemas de aprendizaje *débiles* (en términos de la precisión de sus predicciones), en sistemas *fuertes*. Para ello, mediante la combinación de varios de esos sistemas débiles, y por medio de un proceso iterativo, se conseguía construir un sistema de aprendizaje fuerte, capaz de alcanzar cotas de precisión mucho mayores que las de sus componentes más débiles, y ser por tanto mucho más útil.

Recientemente, los últimos avances en ML y DL han sido originados fundamentalmente por dos motivos:

- Por un lado, la globalización y uso masivo de Internet, que provocó el colosal aumento de la cantidad de datos disponibles, pasando de contar con *datasets* de centenares o miles de ejemplos, durante las últimas décadas del siglo XX, a contar con *datasets* que fácilmente cuentan con millones o decenas de millones de datos, en estos primeros años del siglo XXI. La escasez de datos era una de las razones que impedía, especialmente a algunos modelos de DL, como es el caso de las redes neuronales, no alcanzar todo su potencial, o no conseguir la precisión o eficiencia deseadas, siendo poco prácticas.

- Y, por otro lado, la mejora en la capacidad de computación, propia del avance de la informática y la tecnología en general. Esto a su vez logró reducir los tiempos de entrenamiento para cualquier algoritmo de ML, permitiendo la creación y ejecución de modelos más complejos, y, por tanto, permitiendo enfrentarse a problemas cada vez más difíciles.

2.3. Aprendizaje profundo

En el ámbito de la IA, entendemos como aprendizaje profundo aquella disciplina que aglutina los algoritmos y técnicas, encuadrados dentro del ML, que pretenden modelar abstracciones o representaciones de la realidad, usando arquitecturas computacionales complejas que presentan una estructura jerarquizada y permiten profundizar hasta diversos niveles de detalle. Generalmente, los datos de entrada se representan de forma numérica (matricial o tensorial), y sufren diversas transformaciones a través de las diferentes capas de la jerarquía, partiendo desde abstracciones más simples sobre las cuales se van construyendo otras más complejas en las capas más profundas.

Para hallar uno de los puntos históricos iniciales del campo, debemos remontarnos a Hebb (1949), considerado el padre de la psicobiología, quien, en su libro *The Organization of Behaviour,* presentó una interesante teoría, conocida hoy en día como teoría hebbiana. En ella se centra en la excitación a nivel de neurona, y la comunicación entre neuronas, como tema principal, describiendo un mecanismo de plasticidad sináptica. Según este mecanismo, el valor o la fuerza de una conexión sináptica (conexión entre neuronas) se ve incrementado si las neuronas de ambos lados de la conexión se ven activadas de forma simultánea frecuentemente. Es una idea que informalmente se puede sintetizar como:

"*Las células que se disparan juntas permanecen conectadas*" (Hebb, 1949, p.63)

Unos años más tarde, Rosenblatt (1957) combinó los modelos de interacción cerebral a nivel de neurona propuestos por Hebb, con los esfuerzos de Samuel A. L. en materia de ML, para crear el perceptrón. Inicialmente planeado como un sistema físico, y no simplemente software, el perceptrón ideado por Rosenblatt fue instalado en una máquina construida a medida, conocida como el *Mark 1 Perceptron*, concebida principalmente para el reconocimiento de imágenes. Aunque prometedor, y considerado como la primera neuro-computadora, el perceptrón *Mark 1* tuvo algunos problemas que dificultaron sus posteriores desarrollos, como la escasa precisión reconociendo varios tipos de patrones visuales, como por ejemplo caras humanas o animales.

Más de diez años después aparecerían Minsky & Papert (1969), quienes plantearon la posibilidad de combinar varios de los perceptrones ideados por Rosenblatt, para resolver ciertos problemas, mejorando la potencia de procesamiento ofrecida por el perceptrón simple. Para ello, propusieron que utilizar 2 o más capas en el perceptrón, en lugar de una sola como inicialmente estaba planteado, podía ofrecer mejores resultados. Sin embargo, no disponían de un mecanismo automático para ajustar los pesos de conexión de esas capas intermedias, pues no sería hasta unos años después cuando se popularizarían mecanismos como el de *backpropagation*.

Conviene hablar ahora de un término conocido como *conexionismo*, transversal a muchos de los autores de este campo, y cuyos orígenes se pueden encontrar más allá del último siglo. Se conoce como conexionismo a un enfoque, dentro de la IA, que intenta explicar los fenómenos de la mente como sucesos que emergen de redes complejas formadas por unidades más simples interconectadas. Es decir, no serían las neuronas como tal las que albergasen o produjesen dichos fenómenos, sino su interconexión formando una red compleja la que permitiría que dichos sucesos surgiesen a partir de ella.

Sus bases teóricas pueden remontarse hasta Santiago Ramón y Cajal, considerado también padre de la neurociencia, quien a finales del siglo XIX realizó diversas descripciones de la estructura de las neuronas y sus formas de interconexión, en una de sus ideas fundamentales, conocida como *doctrina de la neurona*. También dentro de este paradigma conexionista, podemos recuperar ideas como la neurona de McCulloch-Pitts (1943), u otros ya mencionados como D. Hebb, con su teoría hebbiana (1949), F. Rosenblatt (1957) y su perceptrón, o los recién citados Minsky & Papert (1969), con la aplicación multicapa del perceptrón de Rosenblatt. Sin embargo, no sería hasta la década de 1980 cuando el conexionismo, como paradigma o disciplina, gozaría de una popularidad notable entre la comunidad científica y tecnológica, principalmente por la aparición de nuevas técnicas.

Entre ellas, cabe resaltar el concepto de retropropagación o *backpropagation*. Realmente se trata de una técnica descubierta y redescubierta en varias ocasiones, y gracias a múltiples investigadores, a lo largo de la década de 1960. Algunos hechos destacables fueron la primera implementación del algoritmo

de forma funcional y computable, gracias a Linnainmaa (1970), quien no pensaba en él como una aplicación para las redes neuronales, sino como un mecanismo de Diferenciación (matemática) Automática (*AD*). O el trabajo de Werbos (1974), quien fue el primero en describir el proceso de entrenar una red neuronal artificial utilizando la técnica de retropropagación de errores.

Por otro lado, no fue hasta Rumelhart et al. (1986) cuando se popularizó realmente la técnica, pues demostraron, experimentalmente, que el método podía ser aplicado para generar representaciones internas útiles de los datos de entrada suministrados a las capas ocultas de redes neuronales profundas. En resumen, con esta técnica se conseguía lo que Minsky & Papert necesitaban para su perceptrón multicapa, una manera de adaptar automáticamente los pesos de las conexiones entre neuronas, en este caso propagando los errores hacia atrás en la red, consiguiendo una forma efectiva de entrenar redes neuronales, la primera forma realmente efectiva, de hecho.

Finalmente, después de esta época se produjo el período mencionado previamente como *invierno de la IA*, que también afectó, como es lógico, al campo del DL. Y tras ello, el ya mencionado resurgimiento de la IA y el ML, gracias al aumento de la cantidad de datos disponibles, algo fundamental para modelos de redes neuronales, del tipo que sean, así como el incremento en la capacidad de cómputo, dando cabida a modelos de redes más complejas, que enfrentaban a su vez problemas más complicados.

Especialmente en DL, a partir de la más reciente década de 2010, se vivió un gigantesco avance para la investigación, si cabe más pronunciado que los anteriores, gracias al uso y popularización de sistemas de computación basados en GPU, o incluso TPU. Estos sistemas aprovechan el paralelismo intrínseco que ofrece este hardware para acelerar de forma notable los procesos de construcción y utilización de redes, facilitando y acelerando, por ejemplo, operaciones matriciales, fundamentales en la fase de entrenamiento de redes neuronales.

Esto ha permitido ofrecer resultados más que destacados en dominios tan diversos como reconocimiento del habla, visión artificial, procesamiento del lenguaje natural, sistemas de recomendación personalizados, sistemas predictivos (valores en bolsa, gestión de recursos, clima) … Citando un par de ejemplos, encontramos a Fayek et al. (2017) y Deng et al. (2013), ambos con modelos para el reconocimiento del lenguaje natural.

2.3.1. Aprendizaje de representación

Antes de una breve introducción a las redes neuronales profundas, conviene mencionar otro término conocido como Aprendizaje de Representación (*RL*), el cual se puede considerar como un paso intermedio entre el ML clásico, y las técnicas específicas de DL. La mejor forma de entenderlos y comprender sus diferencias puede ser mediante una comparativa como la de la Figura 4.

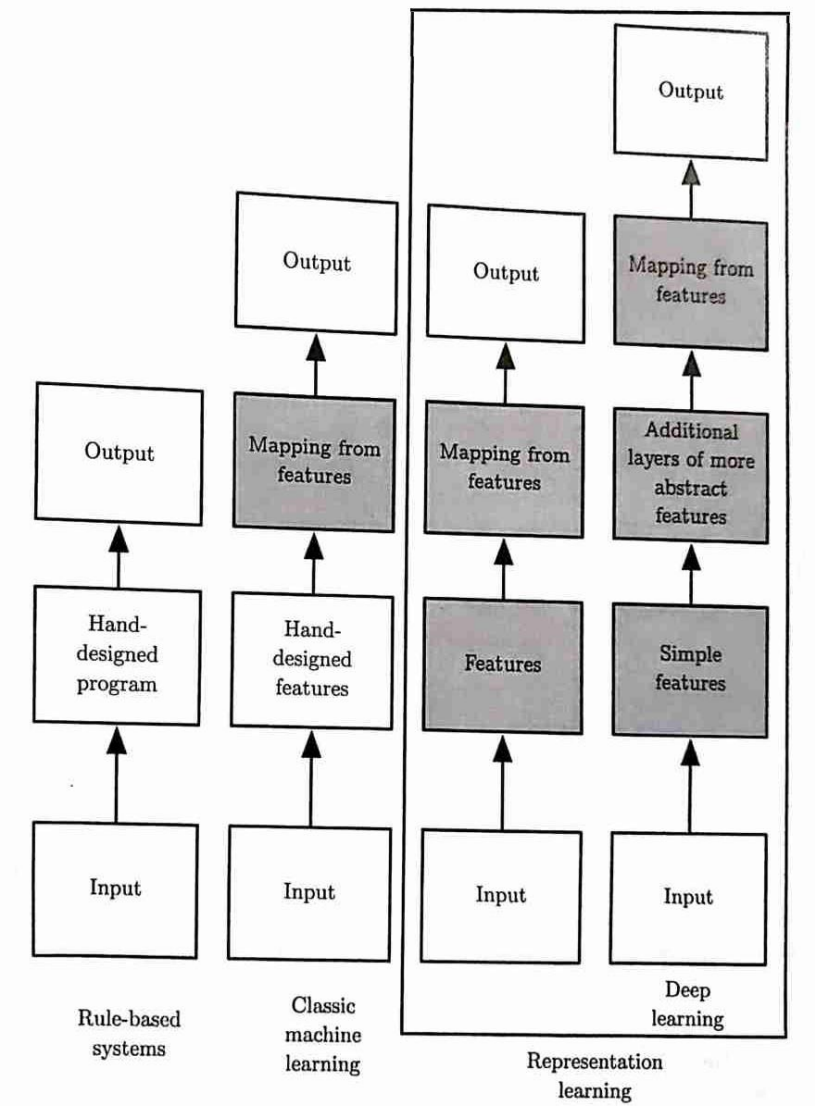

Figura 4. Jerarquía de sistemas de aprendizaje.
Fuente: MC.AI (2018)

En ella se puede ver una jerarquía de sistemas de aprendizaje máquina, en la cual, según avanzamos hacia la derecha, cada sistema es una evolución de su predecesor. Se empieza por los sistemas basados en reglas, donde las normas de operación del software están fijadas de antemano. Es decir, se refiere a los programas informáticos tradicionales.

Por otro lado, tendríamos los ya mencionados sistemas de aprendizaje automático, donde las características a aprender o analizar por parte del software, también se fijan manualmente. En ellos, el proceso de aprendizaje consiste en que el sistema aprenda las relaciones existentes entre dichas características y las salidas del programa deseadas, mediante diversas iteraciones que le permiten ir perfeccionando el ajuste de sus parámetros.

A continuación, tendríamos los recién mencionados sistemas de RL, en los cuales, el sistema de aprendizaje automático no solo aprende la solución al problema a partir de una representación dada en

forma de características. Además, también es capaz de aprender la representación en sí misma, es decir, partiendo de los datos en bruto, aprende cuáles son las mejores características en las que fijarse, acorde al objetivo deseado de un problema específico.

Por último, tenemos los sistemas de DL, los cuales no dejan de ser sistemas de RL, pero cuyo objetivo es ir un paso más allá, y conseguir que el software sea capaz de construir conceptos cada vez más complejos. No solo aprenden la relación entre la solución al problema y las características relevantes de los datos, además de aprender la representación más óptima de esas características, sino que, gracias a la *profundidad* de sus capas, consiguen ofrecer una mayor segregación de conceptos. Así, mientras que en un sistema de RL se puede considerar que las características poseen complejidad equivalente, en el caso de los sistemas de DL, se comienza aprendiendo conceptos simples en sus primeras capas, para luego apoyarse en ellos en las siguientes, en favor de construir otros cada vez más complejos.

2.3.2. Redes neuronales profundas

Se realiza aquí un pequeño inciso teórico, acerca de una de las herramientas principales del aprendizaje profundo. Si bien existen otras técnicas dentro del campo, como el caso de los *autoencoders* o las *redes de creencia profundas*, sin duda el modelo más ampliamente utilizado es el de **red neuronal profunda**. Partiendo de la idea de perceptrón propuesta por Rosenblatt, y más tarde de la combinación de varios de estos en capas, como así lo planteaban Minsky y Papert, nacería el perceptrón multicapa, también conocido como red neuronal.

Una red neuronal, proponiendo una definición más actual y formal, podría definirse como un modelo de función matemática que calcula una salida a partir de una entrada, cuya estructura se define mediante una serie de nodos (neuronas) dispuestos en diferentes capas. Así, cada nodo representa una función más simple, dependiente de los nodos de capas anteriores. De hecho, diversos autores, como Cybenko (1989) o Park & Sandberg (1991), han demostrado de forma teórica que el perceptrón multicapa es un *Aproximador Universal*, debido a que cualquier función matemática continua puede aproximarse utilizando una red neuronal con al menos una capa oculta.

La peculiaridad de considerar a estas redes como profundas se puede enfocar desde dos vertientes. Por un lado, un número elevado de capas, puesto que, a más capas, más profundidad tendrá la red. Por otro lado, esa misma profundidad de capas es la que dota a la red de una jerarquía más profunda de conceptos que pueda representar. Generalmente, lo que se pretende es que los modelos definidos sean capaces de construir y representar de manera automática conceptos cada vez más complejos en las capas ocultas finales, a partir de los conceptos más sencillos que aprenden en las iniciales.

Se muestra en la Figura 5 una visión resumida de la arquitectura de una red neuronal profunda genérica. En ella se puede ver no solo la organización estática de neuronas en capas, sino también su funcionamiento desde un punto de vista dinámico, en sus fases de entrenamiento y predicción.

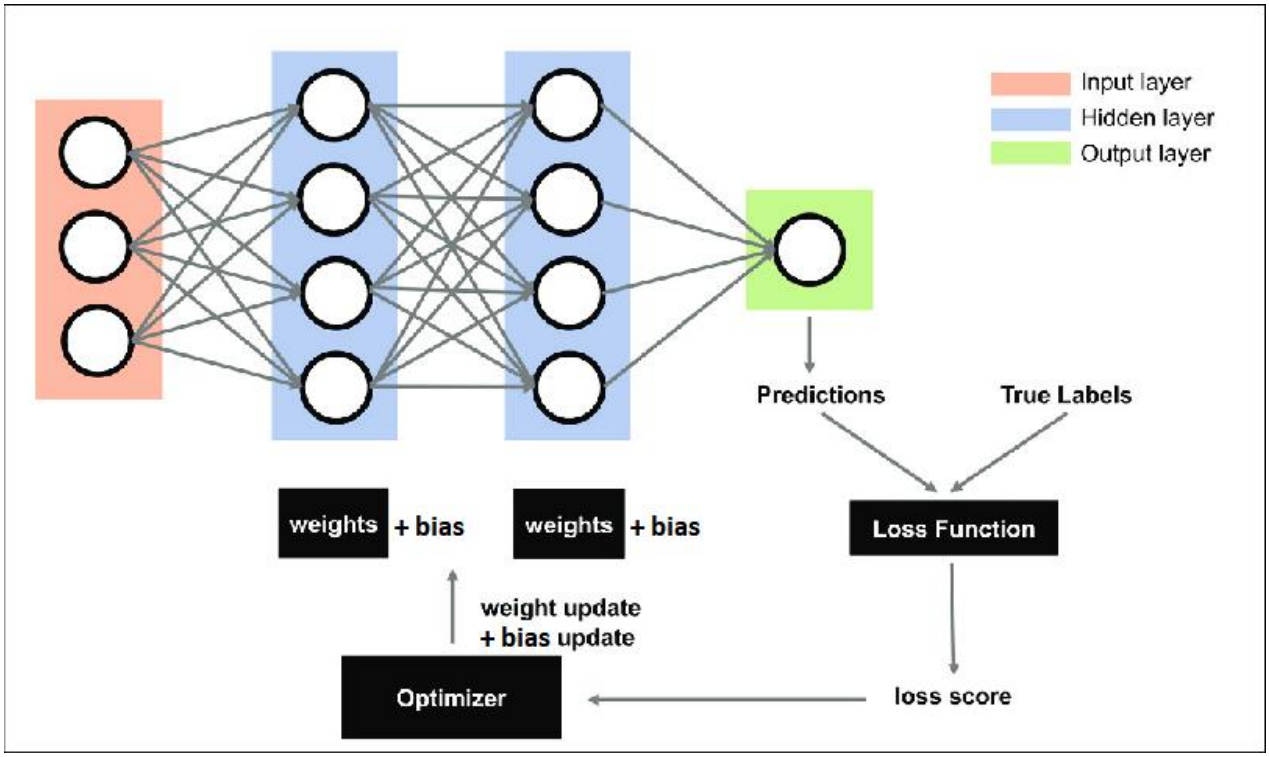

Figura 5. Arquitectura general de una red neuronal profunda.
Fuente: Korkmaz (2019)

Desde el punto de vista meramente estático, se pueden ver hasta un total de 4 capas, siendo la primera (rojo) la capa de entrada, las dos intermedias (azul) sendas capas ocultas, y la capa final (verde), compuesta en este caso por una sola neurona, la capa de salida resultado de los cálculos de la red, y la encargada de obtener las predicciones.

Asimismo, vemos como las capas ocultas, así como la de salida, cuentan con una serie de parámetros conocidos como **pesos**, los cuales se utilizan para ponderar las conexiones entre las neuronas de diferentes capas, incrementando o decrementando la importancia de cada una de esas conexiones en el resultado final. Además de estos pesos, se deben tener en cuenta otros parámetros adicionales, como son los **bias**, que permiten 'desplazar' el resultado para ajustarlo mejor a los datos.

Se introducen aquí dos conceptos mencionados previamente, como son las fases de ejecución de una red neuronal, donde podemos encontrar dos principales:

- **Fase de entrenamiento**. Se trata de la etapa fundamental de cualquier proceso de ML, durante la cual se identifican y aprenden los conceptos o patrones que es posible extraer de los datos en crudo.

Una vez detectados dichos patrones, el conocimiento adquirido, en forma de modelo entrenado, se puede emplear para la siguiente fase.

El funcionamiento general de la red, durante esta fase, es el siguiente:

- Se inicializan los parámetros de la red (pesos y bias), de diversas formas: al azar, todo ceros, siguiendo una distribución normal de probabilidad para los valores, etc.
- Se comienza a proporcionar ejemplos a la red en forma de datos, y por cada ejemplo:
  - Se realizan los cálculos necesarios a través de toda la red, utilizando como entrada para cada capa la salida de la capa anterior, así como los parámetros de la capa, hasta que se llega a la capa final, la cual obtiene una predicción.
  - Dicha predicción se compara con la salida real (*ground truth*) para ese ejemplo concreto, y mediante una función de pérdida, se estima el error cometido.
  - En base a ese error, el optimizador realiza los cálculos necesarios, generalmente empleando un proceso conocido como SGD, basado en el cálculo de gradientes de una función a minimizar, en este caso, la función de pérdida, intentando minimizar dicha pérdida. Así, obtiene unos nuevos valores para los parámetros de la red en cada capa, los cuales son actualizados en el modelo.
- El proceso continúa hasta que se detiene de forma automática o manual, por diferentes motivos. Por ejemplo, cuando se alcanza una precisión deseada, o cuando se supera cierto límite de tiempo de entrenamiento o de uso de recursos por parte de la red.

- **Fase de predicción**. Es el momento en el que se obtiene el verdadero valor, y se aprovecha todo el potencial del ML. Es decir, se hace un uso real de la red, suministrando un nuevo ejemplo, con una salida desconocida, y se recupera el resultado de la capa final como predicción a utilizar. En resumen, si el ML se trata de utilizar los datos para conseguir respuestas, esta etapa es precisamente aquella en la que se obtienen dichas respuestas respecto al mundo real.

2.3.3. Redes neuronales convolucionales

De forma análoga a la sección anterior, se hace aquí una descripción breve de un tipo especial de redes neuronales profundas, conocidas como redes convolucionales, implementadas y utilizadas ampliamente en todo lo referente al procesamiento automático de imágenes.

Se asemejan a las redes neuronales profundas en que su funcionamiento es, a grandes rasgos, el mismo, tomando valores de entrada, realizando una serie de cálculos con ellos a través de sus diferentes capas, empleando para ello los valores de sus parámetros en cada momento, y finalmente obteniendo una estimación de pérdida a minimizar mediante un algoritmo de optimización concreto. En resumen, se podría decir que su similitud se basa en que en ambos casos se trata de sistemas compuestos por neuronas distribuidas en capas, las cuales se optimizan de forma automática a través del aprendizaje, como recoge O'Shea & Nash (2015).

Desde un punto de vista conceptual, también siguen una filosofía de aprendizaje similar a estas últimas, pues, mientras que las redes profundas forman conceptos complejos a partir de otros más simples, en el caso de las redes convolucionales se va construyendo una representación jerárquica de las características o *features* de la imagen, con unas capas iniciales centradas en los elementos más simples de la imagen (bordes, contornos), y las finales enfocadas en obtener representaciones de más alto nivel (identificación y/o localización de un objeto) a partir de dichos elementos más simples.

Una de sus principales diferencias radica en el tipo de capas ocultas utilizadas para la construcción de la red, contando con capas convolucionales y *max-pooling* como novedades, aunque pudiendo emplear también capas densas (FC), como en el caso de las redes profundas clásicas.

- Capas convolucionales.

  Son las más importantes, y las que dan el propio nombre a este tipo de redes. Se basan en el concepto de convolución (matemática), aplicada sobre la matriz de píxeles que representa la imagen a procesar. De forma general, una convolución no es más que una operación que combina dos funciones, generando una tercera función que expresa la forma en la que la primera se ve afectada por la segunda, en tanto en cuanto a su forma o valores se refiere. La convolución sería tanto el proceso en sí, como el propio resultado de aplicar dicho proceso. Se expone en la Figura 6 un ejemplo del esquema simplificado de una convolución sobre un array de píxeles.

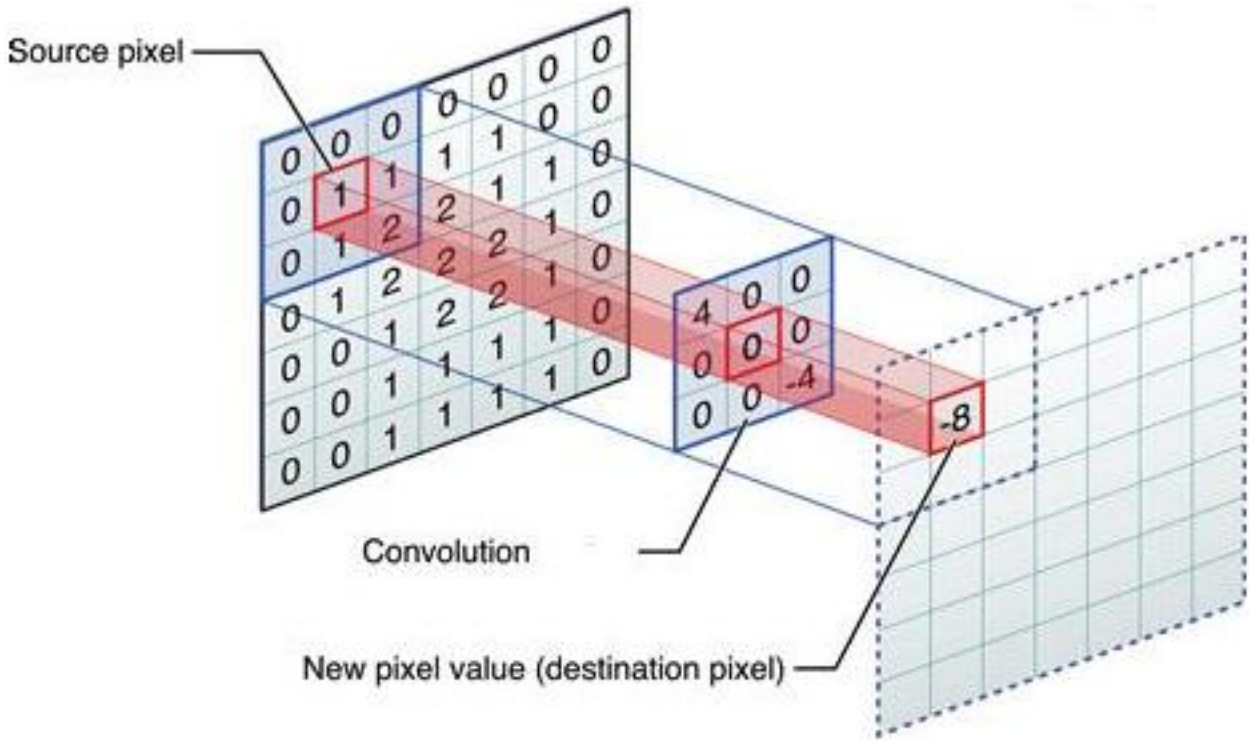

Figura 6. Esquema simplificado de una convolución sobre una matriz de píxeles.
Fuente: Basavarajaiah (2019)

Como se puede ver, en el caso del procesado de imágenes, funcionan a partir de unos filtros n-dimensionales de pequeño tamaño, también conocidos como *kernels*, cuya profundidad, n, es igual a la profundidad de la imagen (por ejemplo, en una imagen RGB, tendrían una profundidad

de tamaño n = 3). Dichos filtros se van desplazando por toda la imagen obteniendo las salidas para esa capa, en forma de un mapa de datos conocido como *mapa de activación* de la capa.

Si nos referimos al ejemplo de la Figura 6, la primera función sería el mapa de píxeles de entrada, la segunda función sería el filtro o *kernel*, y la tercera función, resultado de la convolución, sería el mapa de activación. El proceso centra el *kernel* sobre una región concreta del mapa de entrada, obteniendo como resultado una suma ponderada del píxel central de esa región, así como de los píxeles vecinos abarcados por el tamaño del filtro.

Generalmente, se aplica más de un filtro por cada capa convolucional, con el fin de obtener varias características de la imagen por cada posición de esta, obteniendo así varios mapas de activación como salida de cada capa convolucional. Asimismo, se pueden secuenciar diferentes capas convolucionales, cada una de ellas con diferentes tamaños de filtros, en una jerarquía que siempre tratará de aproximarse de la forma más fiel posible a la granularidad de características presentes en las imágenes de entrada.

- Capas *max-pooling*.

  La idea detrás de este tipo de capas es la de reducir el tamaño de las representaciones obtenidas de manera que se hagan más pequeñas, y, por tanto, más manejables computacionalmente. De esta forma se logra reducir el número de parámetros que tiene que gestionar la red, y permite también prevenir o mitigar el *overfitting*, al limitar el poder expresivo de la red.

2.3.4. Transfer learning

Como apunte breve, conviene mencionar aquí un concepto bastante novedoso, al menos en cuanto a su aplicación a redes neuronales profundas, y es el de *transfer learning*, o transferencia de aprendizaje.

Se conoce como tal (Bozinovski & Fulgosi, 1976) (Bozinovski, 2020) a una técnica y a un campo de investigación propio dentro del aprendizaje automático, que se centra en estudiar el cómo almacenar y aplicar el conocimiento adquirido por un primer modelo o red, sobre un primer problema de aprendizaje, para utilizarlo más tarde como punto de partida con un segundo modelo, sobre un segundo problema de aprendizaje diferente, pero relacionado, logrando de esta forma aprovechar parcialmente los procesos de entrenamiento ya realizados por otros modelos. Se podría decir que, al menos en términos prácticos, consiste en una reutilización de los parámetros de entrenamiento de la red, como los pesos de interconexión de la neuronas, ya calculados previamente para otra tarea.

Por mencionar algunos ejemplos de uso práctico actuales, encontramos: un algoritmo que permite el reconocimiento o verificación de rostros (Cao et al., 2013), empleando *transfer learning*, o un sistema que permite aprovechar el entrenamiento de modelos anteriores para detección automática de desperfectos en el asfalto (Gopalakrishnan et al., 2017).

2.4. Redes neuronales para visión artificial

Una vez realizado un repaso breve por las aportaciones dentro de los campos de CV y ML, incluyendo DL, faltaría mencionar la reciente combinación de ambos campos, que tan buenos resultados está dando actualmente. Se trata del intento de aplicar los modelos de redes neuronales a problemas de visión artificial, consiguiendo mejorar muchos de los resultados obtenidos anteriormente con otras técnicas.

2.4.1. Introducción conceptual e histórica

A nivel arquitectónico, encontramos principalmente dos vías de investigación: las ya mencionadas CNN y las GAN, ambas complementarias.

Por un lado, las CNN surgidas como variación de los perceptrones multicapa. Toman una imagen de entrada y calculan descriptores que expresan características espaciales de esta, con el fin de simplificar la arquitectura de la red, reduciendo en gran medida los parámetros necesarios para entrenarla. Aunque, por otro lado, el cálculo de esos descriptores conlleva una carga de procesado extra. Son ampliamente utilizadas en muchas tareas de CV, como la clasificación de imágenes (Krizhevsky et al., 2012), la recuperación de imágenes similares a partir de un ejemplo, la detección de objetos (Redmon et al., 2016) o la segmentación automática.

En este caso, hablamos de un campo de desarrollo e investigación de reciente aparición, prácticamente surgido o desarrollado durante la última década, pero aun así, podemos encontrar referencias anteriores remontándonos a Fukushima (1979). Él, inspirado igual que otros muchos por los trabajos de Hubel & Wiesel, construyó una red artificial multicapa y jerarquizada, conocida como *Neocognitron*, compuesta por “células”, tanto simples como complejas, que permitía reconocer patrones visuales en imágenes, sin verse afectada por cambios en la posición de estas.

Esta red incluía varias capas convolucionales, las cuales tenían asociados vectores de pesos, conocidos como filtros. Su función era la de desplazarse a lo largo de matrices bidimensionales de valores, como las representadas por imágenes, realizar diversos cálculos en base a la región de la matriz cubierta en cada desplazamiento, y con ello producir eventos de activación (mapas de activación) que servían como entrada para las siguientes capas de la red. Por esto se considera el *Neocognitron* como el “padre” de las primeras CNNs, debido a su evidente similitud.

Una década después, encontramos otro de los hitos en la disciplina, de la mano de LeCun et al. (1989), quienes decidieron aplicar las técnicas de aprendizaje mediante retropropagación a la arquitectura propuesta por Fukushima, construyendo así la primera CNN moderna, conocida como LeNet-5, la cual ya tenía algunos de los elementos fundamentales que se siguen utilizando aún en las CNNs de hoy en

día. De igual manera que ya hiciera Fukushima, LeCun decidió aplicar sus progresos al reconocimiento de caracteres, llegando a crear incluso un producto comercial para leer códigos postales manuscritos.

Además de ello, otro de los logros remarcables del equipo de LeCun fue la creación del *dataset* conocido como MNIST (Yann LeCun et al., 1999), un conjunto de imágenes de dígitos manuscritos, considerado como uno de los benchmarks tradicionales más populares para evaluar la capacidad de diferentes modelos de aprendizaje automático.

Por otro lado, tenemos las GAN, empleadas principalmente en tareas de aprendizaje no supervisado, donde se implementan dos sistemas de redes neuronales que compiten entre sí en un juego de suma cero. Este tipo de redes son principalmente generadoras de contenido de todo tipo, dado que mientras que una de las redes crea nuevos elementos (por ejemplo, imágenes), la otra red evalúa el resultado generado por la primera, basándose en características aprendidas previamente a partir de ejemplos del elemento objetivo. Es por ello por lo que se aplica en áreas como la generación automática de textos, así como la generación de imágenes de todo tipo, manifestando ciertas dotes creativas.

Nacidas, hace menos de una década, de la mano de Goodfellow et al. (2014). Con este nuevo enfoque dentro del campo de DL, se pretendía demostrar que se podía automatizar, a través de una red neuronal, el proceso de evaluar cualitativa y cuantitativamente muestras generadas por otra red neuronal distinta, a través de un proceso de confrontación entre ambas. Se estaban comenzando a explorar caminos antes no abordados en la IA, como el de la generación de contenido, inventiva y originalidad.

Para ello, entrenaban simultáneamente dos modelos: uno generativo (G), encargado de aprender características de un conjunto de datos de entrenamiento y generar nuevos ejemplos en base a ellas, y otro discriminativo (D), que se ocupaba de estimar la probabilidad de que un ejemplo concreto proporcionado perteneciera al conjunto de entrenamiento dado, o bien hubiera sido generado por el modelo G. Se puede entender también como un juego *minimax* de 2 adversarios. Era por tanto el objetivo de G el de generar muestras lo suficientemente próximas al objetivo deseado, con el fin de conseguir engañar a D, mediante una prueba que recuerda mucho a un “test de Turing” entre redes neuronales.

2.4.2. Aplicaciones recientes

Surgiría por aquel entonces, remontándonos hasta 2010 o incluso la década anterior, una verdadera explosión de las redes neuronales, especialmente CNN y GAN, motivada por causas similares a las que propiciaron los notables progresos en materia de DL conseguidos desde principios de siglo:

- Por un lado, una potencia y rapidez de cálculo significativamente mayor, al menos en comparación con 1990, año en el que se lanzó LeNet-5. Al progreso natural de evolución del hardware (Ley de Moore - Tardi (2021)) se sumó el hecho de que se empezaron a usar GPUs y TPUs en DL, logrando significativos avances.

- Por otro lado, la capacidad de acceso a enormes *datasets* de imágenes etiquetadas, como el de ImageNet, gracias a los cuales los investigadores podían entrenar sus modelos de DL lo suficientemente bien para evitar problemas como el *overfitting*.

Muestras de ese fenómeno hay muchas, pero por centrar y mencionar algunas, encontramos:

- **Cuidado de la salud**, para la asistencia en diagnósticos, permitiendo diagnosticar a los pacientes de forma más fácil y fiable. Con algoritmos de CV que pueden ayudar a identificar y detectar enfermedades en fases tempranas, a través de radiografías en las que se puedan detectar patrones comunes a esa enfermedad, por ejemplo.

  A su vez, también dentro del sector de la salud, encontramos software asistencial para operaciones quirúrgicas, utilizando conceptos de CV, así como de realidad aumentada, para guiar con una mayor precisión, y nutrir de suficiente información acerca del paciente a los profesionales que realizan la intervención.

  Podemos observar, en la Figura 7, un ejemplo claro de lo comentado, donde se muestra una serie de mamografías, y, resaltado en color rojo, algunas anomalías o pequeñas irregularidades identificadas respecto a una muestra de un paciente sano. El detectar estos detalles, especialmente si se automatiza y se consiguen resultados en las fases más tempranas de la enfermedad, puede marcar diferencias significativas en enfermedades graves o muy graves, como el cáncer. Y no solo puede ser especialmente útil en fases precoces, sino también pueden servir como segundos o terceros diagnósticos, siempre complementarios a las decisiones médicas tomadas por profesionales humanos, que eviten tratamientos o pruebas en pacientes a los que inicialmente se les habían diagnosticado falsos positivos.

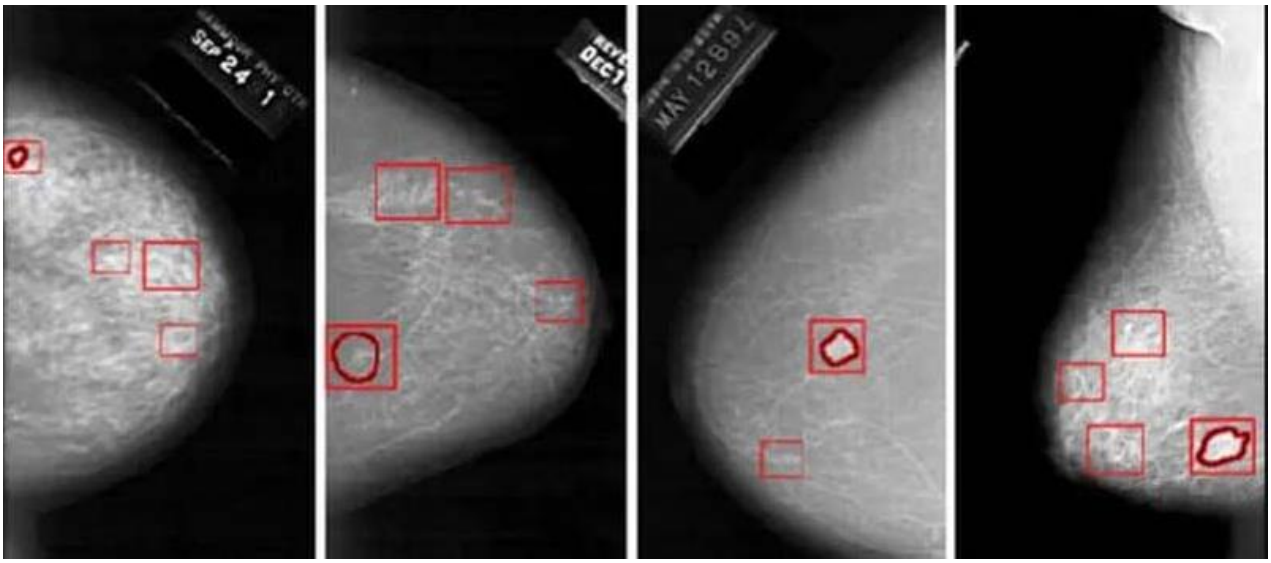

Figura 7. Ejemplo de aplicación médica utilizando CV.
Fuente: Cogito (2019)

- **Comercio**, permitiendo la creación de tiendas automatizadas o desatendidas. Como ejemplo Amazon, que ya ha creado diversos establecimientos con un sistema de CV que permite identificar y seguir a varios clientes de forma individualizada por la tienda, de tal forma que recolecte información acerca de qué productos selecciona cada uno, para vincularlos con su cuenta de manera automática, y facturarles el importe.

Dentro del mismo sector, podemos encontrar también soluciones que permiten realizar un inventariado automático de productos, con robots móviles, equipados con un software de CV capaz de detectar productos fuera de los estantes adecuados, productos dañados o con la etiqueta no reconocible, así como identificar qué productos se han acabado o requieren la reposición de más unidades.

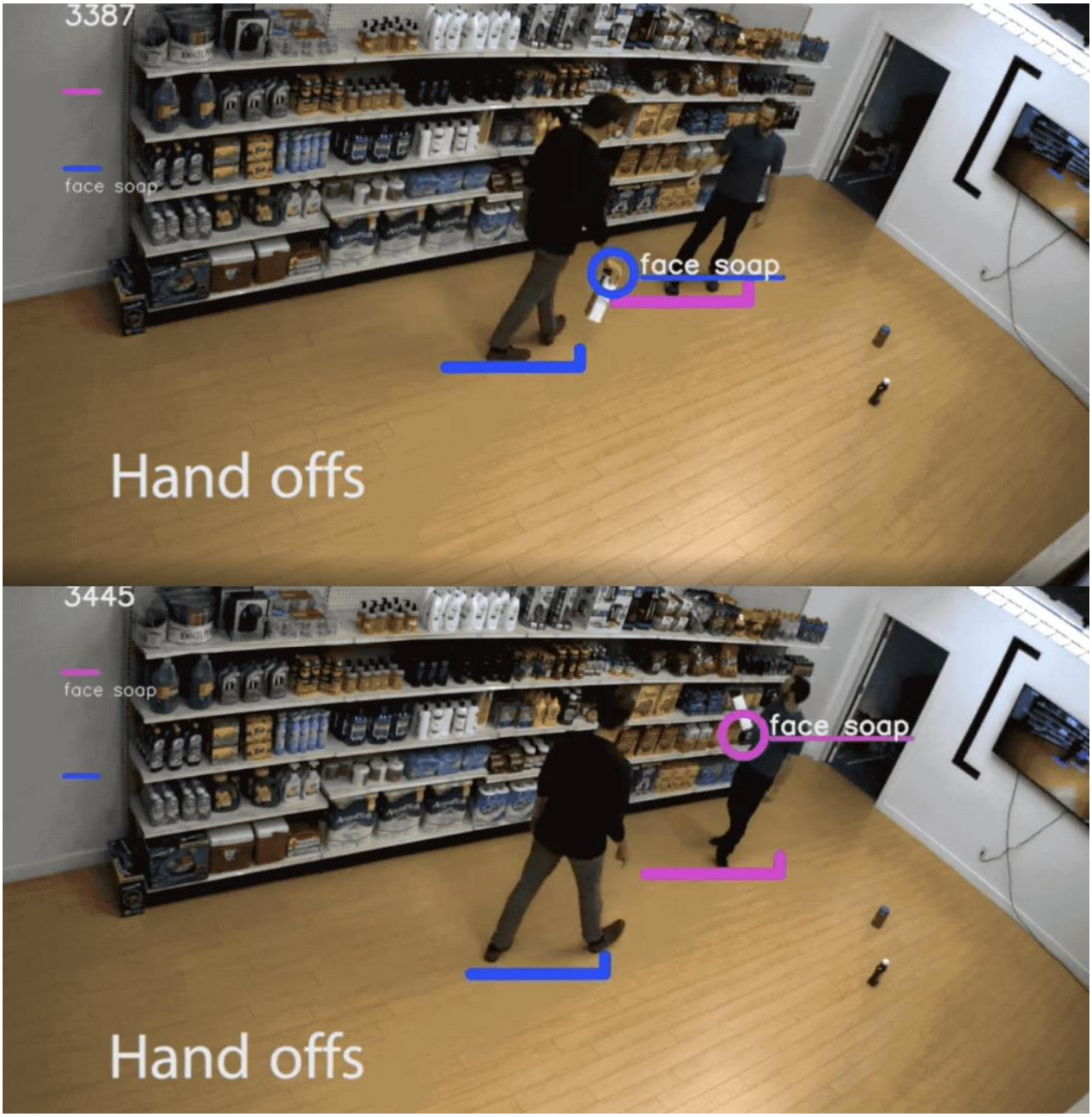

Figura 8. Ejemplo de aplicación de CV en un establecimiento.
Fuente: Captain et al. (2018)

En este caso, se expone la situación de la Figura 8, en la que se puede ver cómo, mediante visión artificial, se puede realizar un seguimiento, en tiempo real, de las personas y productos existentes en el interior de un establecimiento, identificando qué productos están siendo manipulados, así como qué persona es quién está en posesión de dicho producto.

- **Conducción autónoma**, con empresas como Tesla y Waymo a la cabeza, una de las áreas de investigación con mayor potencial actualmente. Las herramientas de CV están ayudando en tareas como la detección de fatiga en el conductor, la ayuda para mantener el carril, la capacidad de mantener la distancia de seguridad entre vehículos...

  Y hablando de aplicaciones más avanzadas, se están desarrollando y probando soluciones que permiten leer y entender señalizaciones viales, detectar peatones y otros peligros ante los que actuar, o apartarse automáticamente ante la llegada de vehículos de emergencia, por ejemplo.

  Bastante conocidas son las imágenes compartidas por parte de empresas como Tesla, como la que aparece a continuación en la Figura 9, donde se muestra y detalla el sistema de CV utilizado en todos sus vehículos autónomos. En dicha figura se pueden apreciar algunos de los conceptos ya mencionados previamente, y podemos hablar, por ejemplo, de cómo el sistema está realizando varias tareas de CV simultáneamente, como es una segmentación por colores, para discernir entre las diferentes clases de objetos que puede detectar, así como las diferentes instancias de esas clases. También, empleando las formas geométricas que rodean dichos objetos, da información precisa acerca de la ubicación de los mismos.

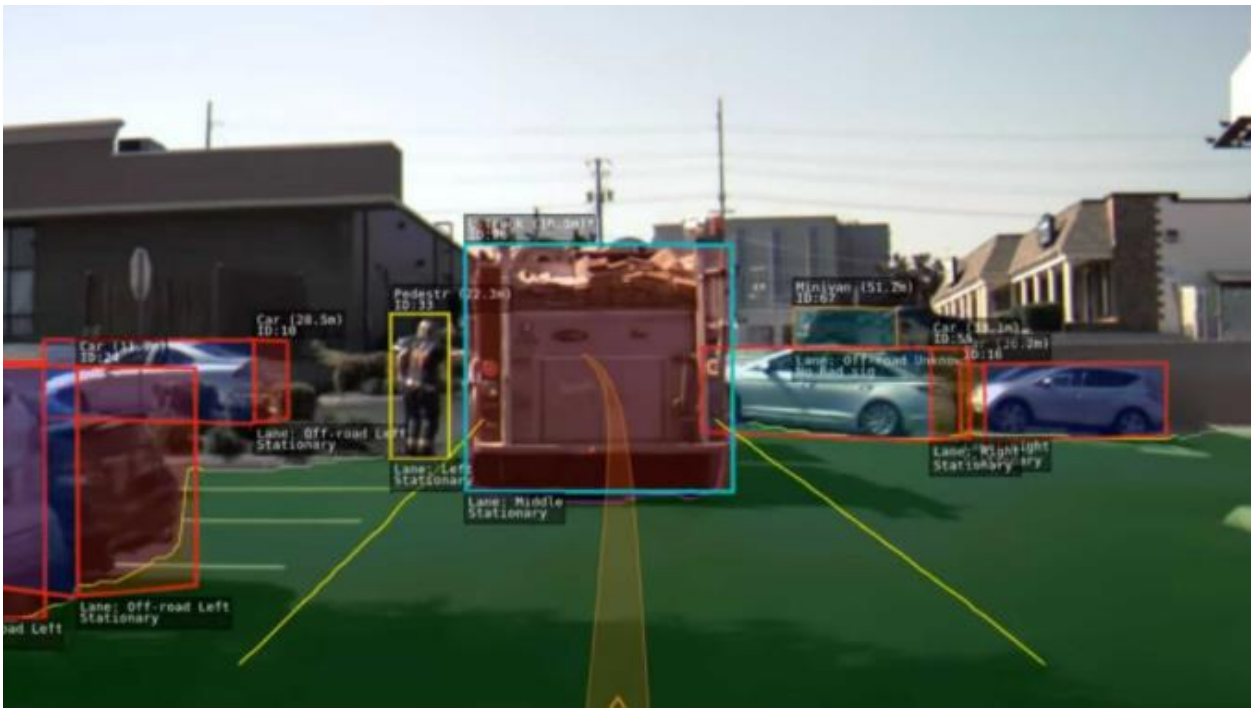

Figura 9. Ejemplo de sistema de CV para un vehículo autónomo (Tesla).
Fuente: Cohen (2021)

- **Marketing mejorado**, donde encontramos empresas tan dispares como Pinterest o Disney.

  El primero, desarrollador de un sistema conocido como *visual search*, el cual se puede ver en la Figura 10. Dicho sistema permite, con una simple fotografía del producto real, obtener una gran cantidad de información del mismo, así como imágenes de otros productos similares, o incluso, enlaces de compra que permiten adquirirlo de forma cómoda y directa.

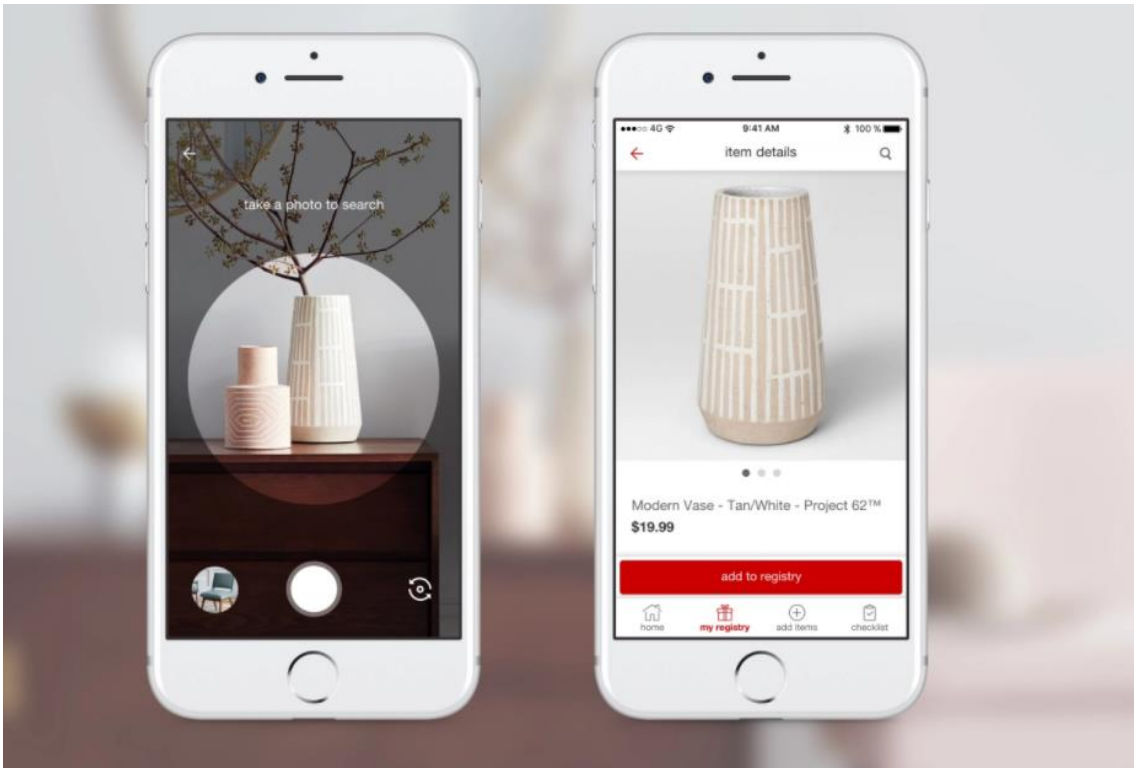

Figura 10. Ejemplo de marketing mejorado, utilizando técnicas de CV.
Fuente: Perez (2017)

El segundo, mediante un sistema que permite capturar las reacciones de los espectadores que acuden a ver sus películas. Esto le ayuda a entender qué escenas de la película provocan qué reacciones concretas, consiguiendo de esta forma conocer de una forma muy precisa las tendencias o gustos del público, y pudiendo desarrollar publicidad mucho más afín a cada nicho de mercado, segmentando incluso los datos por nacionalidades, edades, sexos, etc.

Entre todos estos ejemplos, vemos que hay puntos en común, tareas requeridas para muchas de las soluciones utilizadas hoy en día. Ya se han mencionado alguna de ellas, pero por ampliar y enumerar un listado más completo, encontramos las siguientes, con respectivos ejemplos de aportación:

- Clasificación de imágenes: Krizhevsky et al. (2012)
- Detección de objetos: Redmon et al. (2016)
- Segmentación de objetos: LaLonde & Bagci (2018)
- Seguimiento de objetos en video: Weng et al. (2006)
- Transferencia de estilo en imágenes: Gatys et al. (2016)
- Colorización y aumento de resolución de imágenes: Zhang et al. (2016)
- Generación de imágenes a partir de texto: Qiao et al. (2019)
- Descripción textual de imágenes: Mitchell et al. (2012)

2.4.3. Arquitecturas de redes convolucionales profundas

Años después del *Pascal VOC project*, y siguiendo sus pasos, nacía la iniciativa conocida como ILSVRC, de la mano de Russakovsky et al. (2015), la cual también derivaría en una competición anual, que comenzaría en 2010, y seguiría activa hasta 2017. Mientras que *Pascal VOC* contenía, ya en sus últimas ediciones, apenas 20.000 imágenes a clasificar en tan solo 20 categorías o clases, la primera edición de la ILSVRC disponía de 200.000 ejemplos para el entrenamiento, a clasificar hasta en 1.000 categorías de objetos diferentes.

Dos de las tareas principales de la competición eran la **clasificación de imágenes**, que consistía en asignar una etiqueta de clase a una imagen basándose en el elemento principal de la imagen, y la **detección de objetos**, que englobaba localizar el objeto (u objetos) en la imagen y encuadrarlo, además de identificar su categoría. Dichas tareas se mantuvieron en todas las ediciones celebradas, añadiendo, según el año, alguna extra, como la detección de objetos en video, y no solo imágenes estáticas, o la interesante clasificación de escenas en fotografías, mediante la cual se debía categorizar una fotografía empleando un *dataset* con más de 10 millones de ejemplares, entre un total de 400 clases predefinidas.

A continuación, se presentan algunos de los participantes en las ediciones de la ILSVRC, explicando las características de los trabajos presentados, así como las novedades aportadas por estos. Durante 2010 y 2011, la tasa de error en la clasificación de imágenes se situó en el 28% y 26% respectivamente. Pero no fue hasta 2012, cuando entró a competir la primera CNN, que la tasa de error se reduciría considerablemente, alcanzando el 16%. Desde entonces, la tasa de error fue decreciendo año tras año, hasta alcanzar un destacable 2.3% en el año 2017, y los ganadores, desde el año 2012, siempre fueron modelos de CNNs modernas.

Se describen aquí varias de estas arquitecturas, porque muestran una visión muy completa del estado del arte sobre esta temática, y evidencian también la clara evolución, así como tendencia futura, que están siguiendo muchos de estos modelos aplicados a problemas de visión artificial.

ALEXNET (2012)

Se trataría de la primera red CNN en ganar la competición, creada de la mano de Krizhevsky et al. (2012). Su estructura se puede apreciar en la Figura 11, formada por 5 capas convolucionales (CONV), 3 capas *max-pooling* (MAXPOOL) y 3 capas *fully-connected* (FC), una de las cuales se trataba de una capa *softmax* de 1.000 clases:

CONV #1 MAXPOOL #2 ||| CONV #3 MAXPOOL #4

||| CONV #5 CONV #6 CONV #7 MAXPOOL #8 ||| FC #9 FC #10 SOFTMAX #11

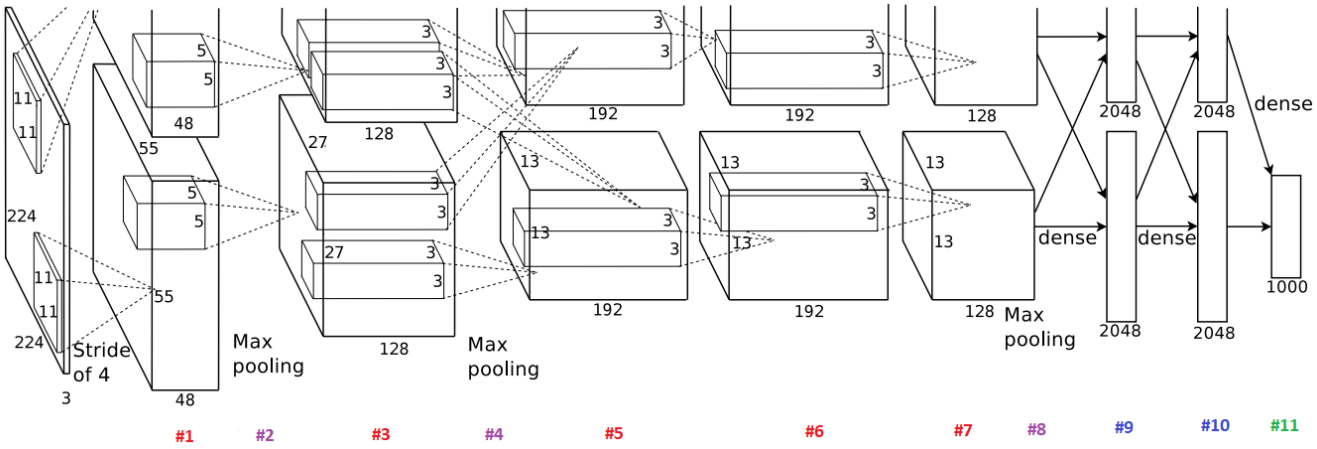

Figura 11. Estructura de AlexNet.
Fuente: Krizhevsky et al. (2012)

Contando con más de 60 millones de parámetros entrenables y 650 mil neuronas, la red se entrenó en 2 GPUs en paralelo durante una semana, pues no era posible almacenar la red completa en una sola, de ahí la segmentación que se ve en el esquema de la figura. Uno de sus descubrimientos importantes fue el uso de *ReLU* como función de activación, en lugar de los usualmente empleados *sigmoid* o *tanh*, acelerando así el proceso de entrenamiento sin perder precisión en el resultado final.

Para reducir el *overfitting*, emplearon 2 técnicas. La primera, conocida como *data augmentation*, consistía en realizar pequeñas perturbaciones aleatorias (cambios leves de iluminación, rotaciones, simetrías…) de las imágenes de un conjunto de entrenamiento, para obtener nuevas imágenes con las que entrenar, con la misma clase que las originales, ampliando de este modo el *dataset* disponible. Y la segunda, *dropout*, una técnica de regularización reciente para la época, pero que resultó ser efectiva y ha pasado a ser muy utilizada hoy en día. Dicha técnica se basaba en aplicar una salida 0 (inactividad) a un porcentaje de neuronas de la red durante el entrenamiento de manera aleatoria, siendo valores muy usados de parametrización 0.25 y 0.5. Con esto se conseguía evitar el peligro de que la red memorizase los datos de entrenamiento, haciendo que le costase más aprender un concepto.

Esta arquitectura supuso un verdadero hito en la competición, así como en el campo de la visión artificial profunda, iniciando un camino de investigación a partir del cual se siguieron desarrollando modelos de CNNs con arquitecturas cada vez más complejas y profundas, mejorando a su vez los resultados.

GoogLeNet (2014)

Siendo ganadora del reto en 2014, la red neuronal presentada por Szegedy et al. (2014) pretendía, en cierta manera, homenajear con su nombre a la red en la cual está basada, *LeNet*, de Yann LeCun, además de hacer referencia a la empresa matriz del desarrollo, pues el equipo de investigación pertenecía al departamento conocido como *Google Research*.

Una de sus características diferenciadoras fue sin duda la aproximación utilizada durante su construcción, pues se apartaba de lo visto hasta ahora, donde se creaban las redes apilando capas de forma secuencial, para pasar a paralelizar algunos procesos. Para ello, se valía de un nuevo tipo de módulo conocido como *Inception*, presentado en la Figura 12.

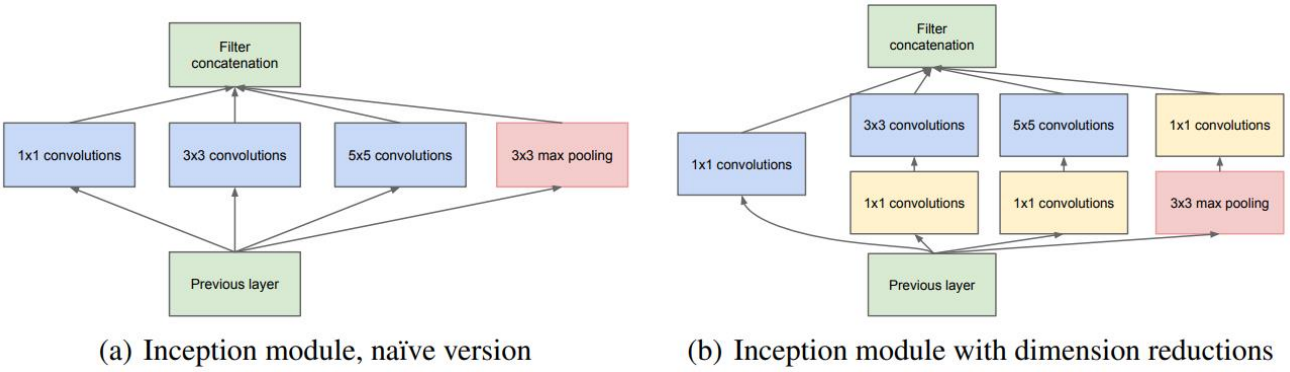

Figura 12. Módulo Inception de GoogLeNet.
Fuente: Szegedy et al. (2014)

Como se puede apreciar, en la parte **a)** de la figura tenemos el módulo *Inception* "ingenuo", o simple, que no incorpora capas de reducción de dimensionalidad, y en la parte **b)** tenemos el módulo *Inception* utilizado en el modelo *GoogLeNet*, el cual si incorpora dichos mecanismos, los cuales se tratan de unas capas de convoluciones 1x1, coloreadas en amarillo, responsables de conseguir un menor tamaño en las dimensiones a procesar, agilizando los cálculos.

El módulo *Inception* conseguía otra ventaja gracias a su arquitectura, y es la paralelización de procesos. Normalmente, después de una capa convolucional, se sitúa una capa de *pooling* o bien otra capa convolucional más, teniendo que decidir el tamaño del filtro en ambos casos. Gracias a este módulo, se podían realizar varias de estas operaciones en paralelo y con distintos tamaños de filtro, pues aplicaba simultáneamente convoluciones de tamaños 1x1, 3x3 y 5x5, así como *pooling*, consiguiendo extraer varias características de la imagen a la vez.

Con la reducción de dimensionalidad, pudieron aumentar la anchura y profundidad de la red final, sin una penalización de rendimiento notoria. Gracias a ello, la arquitectura final de la red contaba con 22 capas, y el número total de parámetros a entrenar se situaba en una cantidad hasta 12 veces inferior a los usados por una de sus predecesoras, la ganadora del año 2012, *AlexNet*, a la par que conseguía una precisión significativamente mayor.

Con esta red se consiguió una precisión bastante cercana al nivel humano, demostrando que había espacio para la creatividad y la innovación, y abriendo la puerta para algunas arquitecturas posteriores, bastante más complejas y profundas.

VGGNET (2014)

Con una tasa de error ligeramente superior a la ganadora de ese año (*GoogLeNet*), encontramos el modelo *VGGNet* de Simonyan & Zisserman (2014), que contaba con hasta 3 variaciones.

Este modelo, igual que el recién mencionado *GoogLeNet*, ayudó a reforzar una idea hasta entonces intuitiva, que era que para que los modelos de CNN consiguieran una comprensión de los datos suficientemente buena, la red debía tener la profundidad suficiente, para ser capaz de representar jerárquicamente, con una granularidad aceptable, los conceptos necesarios. Así, Simonyan & Zisserman presentaron hasta 3 propuestas de modelos de red, con 16, 16 y 19 capas respectivamente, cercano a las 22 planteadas por *GoogLeNet*.

Desgranando la arquitectura más profunda, de 19 capas, nos encontramos con 16 capas convolucionales con filtros 3x3, y 3 capas FC. Todas las capas convolucionales tenían un *stride* (ventana de deslizamiento del filtro) de tamaño 1, y contaban con *ReLU* como función de activación. Además, las capas FC estaban regularizadas mediante *dropout*. Con una estructura bastante más simple que su alternativa ese año, consiguió bastantes buenos resultados, entrenando el modelo en 4 GPUs durante casi 3 semanas, y logrando que incluso generalizase bastante bien empleando otros *datasets*.

RESNET (2015)

La ganadora del año 2015 sería el modelo *ResNet*, de He et al. (2015), basado en bloques residuales, el cual consiguió tasas de error de entre el 3-4%, logrando reducir de forma considerable la tasa de error humana, situada en torno al 5-10%, según la persona.

Una de las ideas principales y revolucionarias de esta arquitectura, fue la de introducir el concepto de bloque residual, como una forma de añadir “atajos” entre capas, de tal forma que se pudieran saltar algunas de ellas. Por entonces, ya estaba demostrado empíricamente que añadir capas a partir de una cierta profundidad de la red, no siempre era motivo de mejora de rendimiento. En realidad, podía ocurrir justamente lo contrario, apareciendo problemas como el *desvanecimiento de gradiente* o la *maldición de dimensionalidad* a medida que se aumentaban las capas.

Se presenta en la Figura 13 el modelo de bloque residual, que según sus autores facilitaba la optimización y convergencia de redes extremadamente profundas. Tanto fue así, que la arquitectura de *ResNet* prácticamente septuplicó lo conseguido anteriormente por *GoogLeNet* o *VGGNet* en relación al número de capas de la red, llegando a conseguir hasta 152 capas de profundidad.

Argumentaban que incrementar el número de capas apiladas no siempre tenía porqué empeorar el rendimiento de la red, porque se podían apilar capas que mapearan la función identidad (a la salida se obtiene la entrada introducida), de tal forma que estas capas, en la práctica, no estuvieran haciendo

ningún cálculo, y por tanto el rendimiento no se viera afectado por la elevada profundidad. Además, durante la fase de *backpropagation*, se evita que el gradiente se desvanezca gracias a las operaciones de suma que permiten ir balanceándolo de forma adecuada.

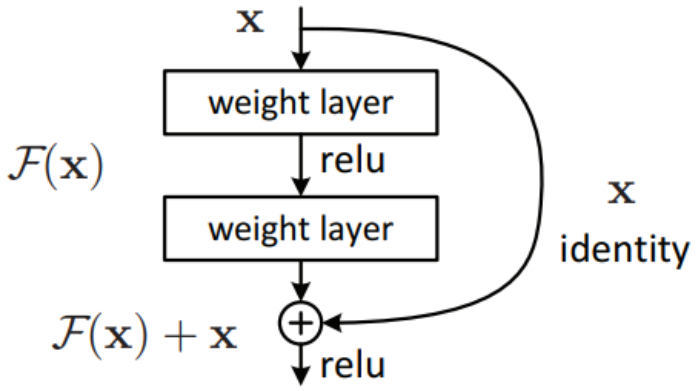

Figura 13. Bloque residual de ResNet.
Fuente: He et al. (2015)

SENET (2017)

Ya en el año 2017, esta vez el premio de la competición se lo llevaría el departamento de Ciencia e Ingeniería de la Universidad de Oxford, Hu et al. (2018), quienes presentarían la arquitectura conocida como *SENet*, llamada así por la nueva construcción de bloques propuesta, *Squeeze-and-Excitation*. Se trataba de un bloque que recalibraba de forma adaptativa los mapas de activación de características resultado de las capas convolucionales, modelando explícitamente las dependencias entre ellas, sin incurrir en un coste computacional añadido significativo. Se muestra el modelo en la Figura 14.

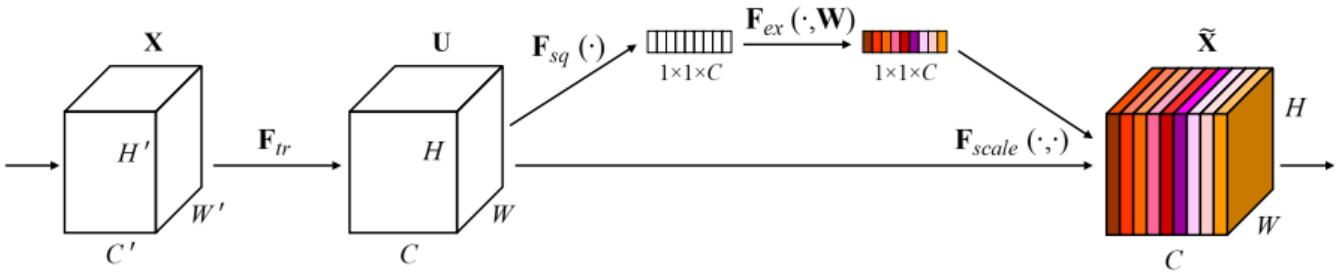

Figura 14. Bloque "Squeeze-and-Excitation" de SENet.
Fuente: Hu et al. (2018)

Como se puede ver, en la fase de *squeezing* se parte de la colección de descriptores de características locales (U), cuya información estadística es importante para la imagen a nivel global, es decir, los mapas de activaciones resultado de las capas convolucionales. Se comprimen dichos mapas en un vector de tamaño N (número de capas convolucionales) mediante una técnica conocida como *global average pooling*, que permite obtener la información estadística mencionada.

Más tarde, durante la fase de *excitation* se aprenden los pesos que modelan las relaciones entre capas, y se utilizan dichos pesos para redimensionar los mapas de activación, ponderando de esta manera cada uno según su relevancia. De esta forma, se consigue una vía para parametrizar (y aprender dichos parámetros) cada paso dado por la CNN.

Gracias a este modelo organizativo de redes, y al nuevo bloque conceptual presentado, consiguieron reducir la tasa de error hasta el 2.3% en el conjunto de datos de validación. Para ello, tuvieron que entrenar el modelo con un tamaño de minibatch de 2048 elementos y SGD sincronizado como optimizador, empleando la potencia de cálculo de hasta 64 GPUs durante un periodo de 20 horas Como se puede ver, los tiempos de entrenamiento del modelo son sensiblemente menores que los de arquitecturas CNN presentadas en años anteriores. Esto se debe a importantes optimizaciones desarrolladas *ad hoc*, tanto en la memoria de las GPUs como en el paso de mensajes entre estas.

2.5. Marco tecnológico actual

Dentro de este capítulo se realiza un rápido repaso por el panorama tecnológico actual, dentro del campo de IA, y especialmente ML, como una ampliación del estado del arte a un nivel más técnico. En él, podemos encontrar y ver competir diferentes librerías, frameworks, datasets o herramientas en general, la mayoría de ellas de muy reciente aparición, que pujan por ser las soluciones más utilizadas en campos como el aprendizaje profundo o la visión artificial. Todas ellas, especialmente las centradas en redes neuronales, presentan **notables ventajas**:

- Simplifican la construcción de modelos complejos para redes neuronales, proporcionando eficientes abstracciones de los procesos a bajo nivel de neuronas y sus operaciones.
- Permiten la automatización de ciertos pasos del funcionamiento de una red neuronal, como es el cálculo de gradientes. Muchos frameworks modernos realizan el *backward* pass de manera automática, de tal forma que el programador solo debe centrarse en diseñar la arquitectura de la red, sin tener que definir ese paso.
- Facilita la ejecución y entrenamiento de las redes neuronales en entornos distribuidos, así como también la compatibilidad con la ejecución del código en GPUs.
- Aumenta la eficiencia y optimización de la construcción, el entrenamiento y la ejecución de redes neuronales, pues estas librerías suelen utilizar implementaciones altamente optimizadas para sus estructuras de datos y algoritmos internos, según su lenguaje base.

2.5.1. Librerías y *frameworks*

Por un lado, si hablamos de frameworks o librerías, encontramos los siguientes ejemplos.

Tensorflow [3]

Conocido por ser uno de los frameworks para DL más utilizados a nivel mundial, se trata de una herramienta *open-source* para computación numérica, que internamente utiliza grafos de computación para la representación y abstracción de los modelos de redes empleadas. Su nombre se debe a que los datos de entrada vienen dados en forma de **tensores** (entidad matemática algebraica) que **fluyen** a través de las distintas operaciones definidas por el grafo de computación diseñado.

Es tanto compatible para ejecutarse en CPUs como en GPUs, y además permite la ejecución en entornos restringidos, como dispositivos móviles, haciéndolo muy flexible. Desarrollado por Google, y usado de forma mayoritaria en muchas empresas, como en el propio departamento de *Google Deepmind*, para tareas de investigación. Ha mantenido desde sus inicios una candente disputa en la industria de frameworks de aprendizaje profundo, en especial con herramientas similares como *PyTorch*.

Keras [4]

Escrita en Python, es una librería de alto nivel, lo que quiere decir que proporciona una interfaz modular y amigable que permite a los desarrolladores definir modelos para redes neuronales de una manera ágil, sin tener que reescribir constantemente el código a bajo nivel de los mismos elementos una y otra vez.

Facilita el desarrollo rápido de prototipos para pruebas, es sencillo de aprender, y funciona directamente sobre las operaciones de bajo nivel de *Tensorflow*, aunque también es compatible con otros frameworks de bajo nivel como *Theano* o *Microsoft CNTK*. Es una de las herramientas preferidas por los desarrolladores que están comenzando a dar sus primeros pasos en el campo del ML.

PyTorch [5]

Se trata de una librería *open source* basada en Python, focalizada en la realización de cálculos numéricos mediante tensores, utilizando el mismo concepto matemático que *Tensorflow*. Esto facilita su aplicación para el desarrollo de redes neuronales profundas, siendo una de las opciones más asequibles para la creación de dichos modelos de aprendizaje profundo.

Desarrollado por Facebook, y caracterizado por su facilidad de uso y capacidad nativa para ejecución en GPUs, se ha convertido en los últimos años en una de las principales opciones por la mayoría de desarrolladores, junto a los propios *Tensorflow* y *Keras*. Como ejemplo tenemos la empresa Tesla, referente en IA y ML, la cual emplea esta librería, entre otras, para construir sus soluciones de mercado.

[3] Página web oficial del ecosistema Tensorflow: https://www.tensorflow.org/
[4] Página web oficial de la librería Keras: https://keras.io/
[5] Página web oficial de la librería PytTorch: https://pytorch.org/

Theano [6]

Cesando su desarrollo en 2017, está librería desarrollada en Python es todavía una plataforma potente y eficiente para el aprendizaje profundo. Permite definir, optimizar y evaluar expresiones matemáticas con matrices de valores de una forma muy eficaz, utilizando de forma transparente la GPU, lo que resulta en operaciones con un alto rendimiento, en comparación a un proceso análogo en la CPU.

Una de sus mayores ventajas es que compila de manera muy optimizada los cálculos que necesita realizar, tanto para CPUs como para GPUs, haciendo su ejecución más eficiente, especialmente en tareas numéricas. Sin embargo, en ocasiones su sintaxis puede resultar de nivel demasiado bajo, por lo que puede requerir de otras librerías para conseguir un nivel de abstracción más elevado.

Microsoft CNTK [7]

Microsoft Cognitive Toolkit, también conocido como *CNTK*, es un conjunto de herramientas *open source* para aprendizaje automático. Usa un grafo dirigido para representar las redes neuronales como una secuencia de pasos computacionales, y está especialmente diseñado para soportar y manejar ingentes modelos y datasets, enfocados en tareas de producción de sectores industriales de gran tamaño.

Dispone de un lenguaje de descripción de modelo, llamado *BrainScript*, el cual también se puede usar de manera independiente. Además, se puede integrar con Python, C++, .NET platform y Java, haciéndolo muy versátil. También es escalable, pudiendo pasar de ejecutarse en CPU, a GPU, hasta en varias máquinas de forma distribuida.

sci-kit learn [8]

Librería, también de código abierto, y también desarrollada en Python, que soporta las distintas fases de varias técnicas de aprendizaje automático, tanto supervisadas como no supervisadas. Es una herramienta con múltiples propósitos dentro de la IA, incluyendo problemas de clasificación, de regresión, de agrupamiento, etc.

Permite, a su vez, la utilización de varias técnicas comúnmente empleadas como:

- La reducción de dimensionalidad, para decrementar el número de variables aleatorias de un problema y poder llevarlo a una simplificación con menor complejidad.
- La selección de modelos, que permite escoger, de entre varios candidatos de modelos de aprendizaje automático, el adecuado para un problema dado y con un dataset concreto.

[6] Página web oficial del proyecto Theano: https://theano-pymc.readthedocs.io/
[7] Página web oficial de Microsoft CNTK: https://docs.microsoft.com/en-us/cognitive-toolkit/
[8] Página web oficial de la librería scikit-learn: https://scikit-learn.org/

- El preprocesamiento de los datos, con el fin de limpiar, normalizar, transformar o seleccionar toda aquella información de entrada al sistema, dando como resultado el dataset final de entrenamiento que se utilizará. Acelera el procesamiento posterior de los datos, evitando errores, y facilitando a su vez el descubrimiento del conocimiento por parte del modelo.

- La validación cruzada, especialmente útil a la hora de entrenar y evaluar modelos, cuenta entre sus principales ventajas con un uso mucho más eficiente del conjunto de datos disponibles, al utilizarlos todos, divididos en subgrupos aleatorios del mismo tamaño, procesables en paralelo.

2.5.2. Datasets

Y en cuanto a datasets, además de los previamente mencionados *ImageNet* y *Pascal VOC Project*, cabe citar algunos nombres, desde dentro del enfoque de la visión artificial.

MNIST [9]

Reseñado brevemente en el repaso histórico de la sección 4 del capítulo 2, redes neuronales para visión artificial, este dataset es considerado una de las piedras angulares en los inicios de los sistemas de aprendizaje de patrones visuales. Formado por un conjunto de entrenamiento de 60 mil muestras, y un conjunto para test de 10 mil, podemos observar algunos ejemplos en la Figura 15.

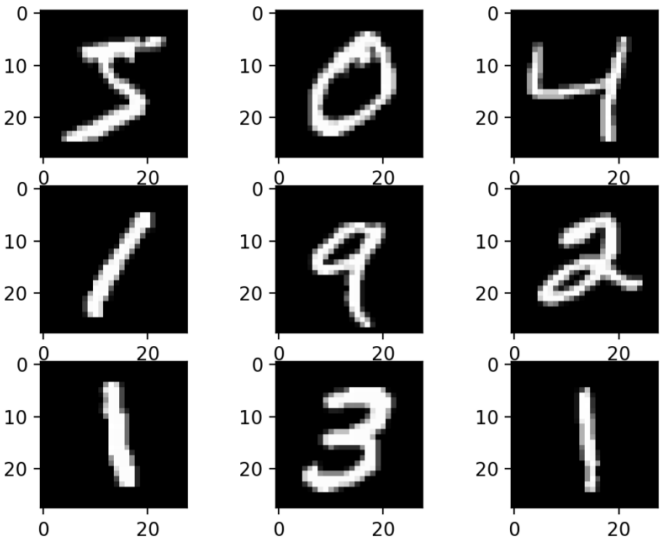

Figura 15. Ejemplos de MNIST.
Fuente: Brownlee (2019)

En ella se puede ver los elementos individuales, imágenes en blanco y negro de dígitos manuscritos de la época, normalizadas a 28x28 píxeles y recopilados uno a uno manualmente.

[9] Página web oficial del dataset MNIST: http://yann.lecun.com/exdb/mnist/

Fashion-MNIST [10]

Formado, al igual que el anterior, por un conjunto de 70 mil imágenes (60 mil para entrenamiento y 10 mil para test) en blanco y negro normalizadas a un tamaño de 28x28 píxeles, este dataset fue creado por la empresa de ropa Zalando, y en este caso no se trata de dígitos manuscritos, sino un conjunto de imágenes de los propios artículos de la tienda, como se puede ver en la Figura 16.

Se puede ver que hay conjuntos de imágenes muy similares, correspondientes a los elementos de una de las 10 categorías en las que se divide el dataset, siendo cada una de ellas un tipo diferente de prenda de ropa: bolsos, chaquetas, camisetas, pantalones, botas, etc.

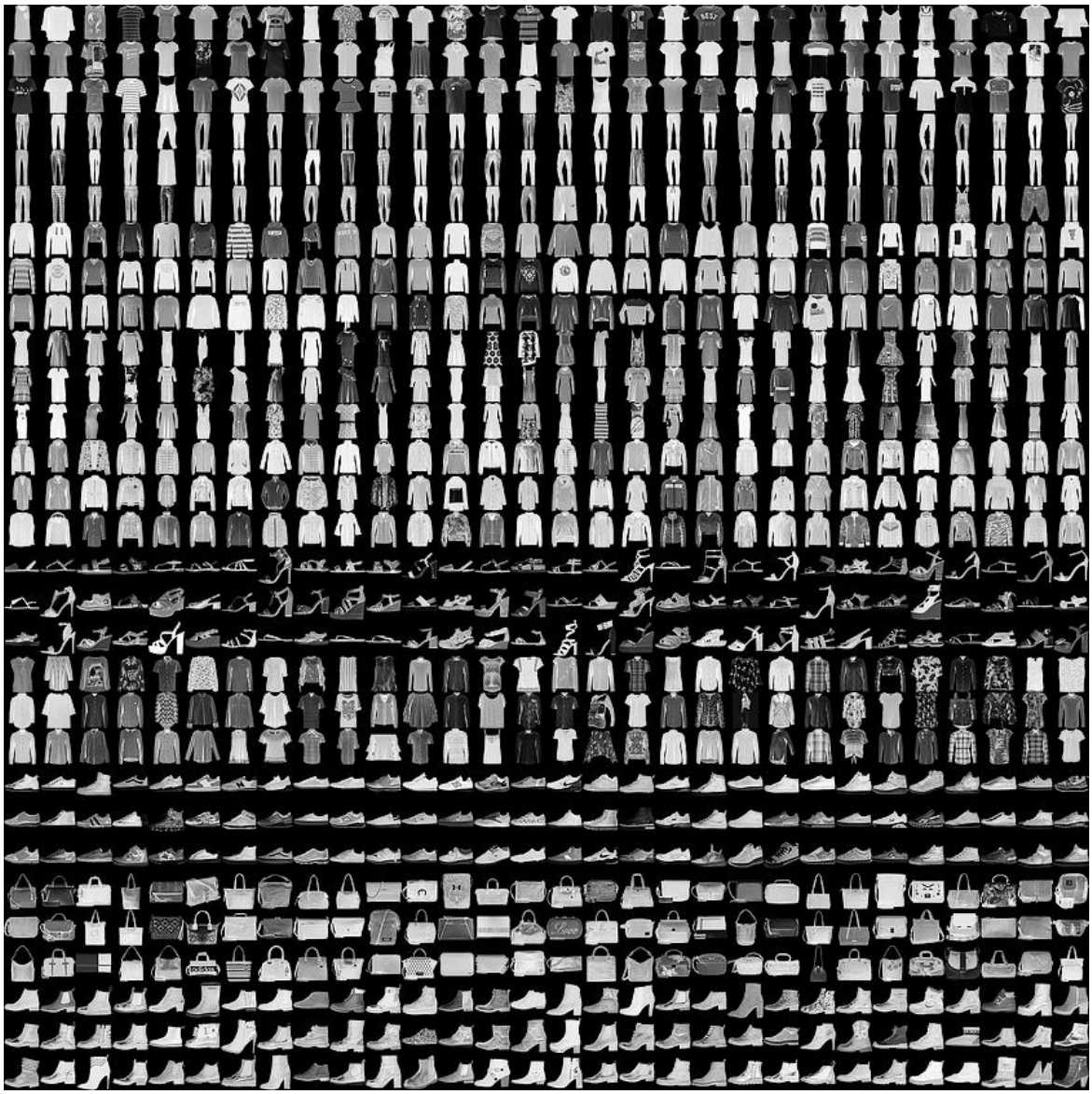

Figura 16. Ejemplos de Fashion MNIST.
Fuente: Rasul (2017)

[10] Página web de Fashion MNIST en la competición Kaggle: https://www.kaggle.com/zalando-research/fashionmnist

CIFAR-10 [11]

Consiste en un conjunto de 60 mil imágenes en color, con un tamaño de 32x32 píxeles, siendo 50 mil para entrenamiento y 10 mil para test. Observemos un ejemplo en la Figura 17.

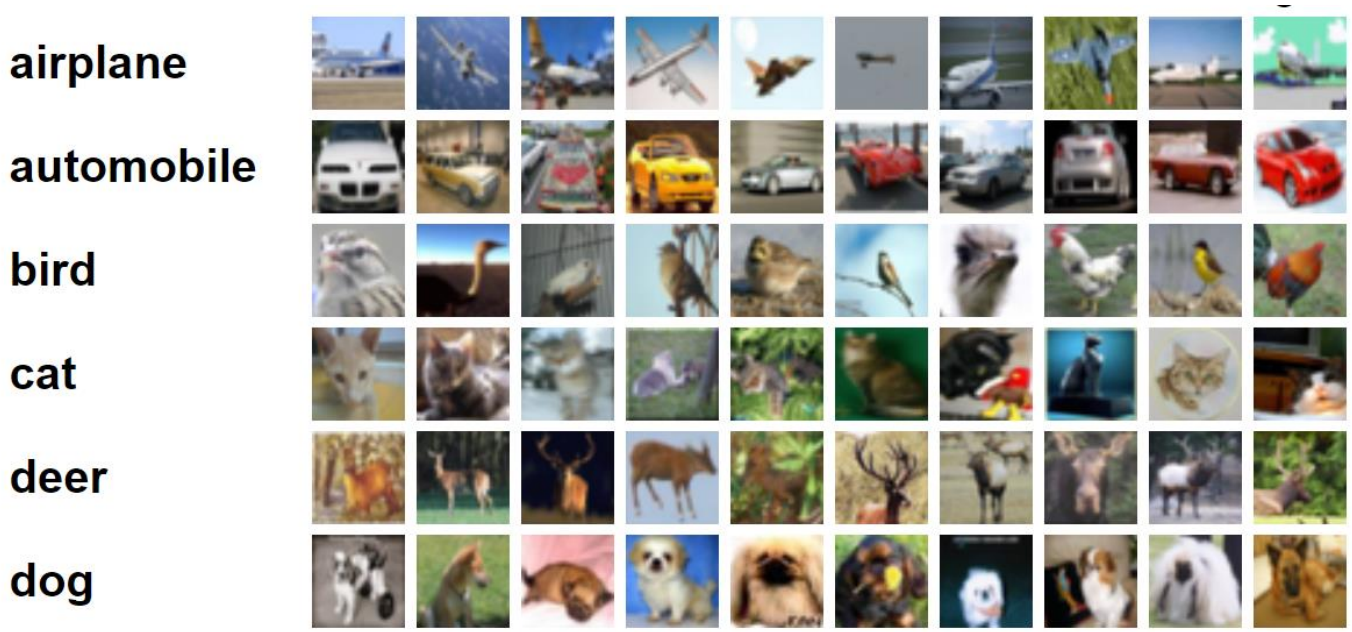

Figura 17. Ejemplos de CIFAR-10.
Fuente: Ardi (2021)

Como se puede ver, se trata de fotografías tomadas del mundo real, clasificadas hasta en 10 categorías diferentes (no se muestran todas en la imagen, por contener su tamaño). Todas las clases son mutuamente exclusivas, no pudiendo haber elementos que pertenezcan, ni siquiera parcialmente, a más de una clase simultáneamente. Utilizado especialmente en tareas de clasificación de imágenes.

Existe también una versión de este mismo dataset, pero con 100 categorías diferentes, conocida como CIFAR-100, la cual no se explica por ser muy similar a la comentada aquí. Una de sus peculiaridades, además de contar con bastantes más clases, es la de que no solo introduce nuevas categorías, sino además supercategorías o superclases, las cuales agrupan varias de las categorías individuales.

COIL-100 [12]

Se trata de un dataset con 100 categorías diferentes de objetos, formado por un total de 7200 ejemplos. Se muestran algunos de ellos en la Figura 18.

Como se puede apreciar, todos ellos están capturados sobre un fondo negro, para facilitar su procesado, y de cada uno de los objetos se tomaron un total de 72 muestras, cada una de ellas separada por 5

[11] Página web oficial del dataset CIFAR (10&100): https://www.cs.toronto.edu/~kriz/cifar.html
[12] Página web oficial del dataset COIL-100: https://www.cs.columbia.edu/CAVE/software/softlib/coil-100.php

grados angulares, rodeando por completo el objeto a lo largo de sus 360 grados. Es de resaltar que los objetos se caracterizan por tener una variedad amplia de formas geométricas y reflejos luminosos.

Figura 18. Ejemplos de COIL-100.
Fuente: El Ghawalby (2015)

xView [13]

Es uno de los datasets públicos disponibles de mayor tamaño que existen, en tanto en cuanto a imágenes aéreas o por satélite se refiere. Cuenta además con una competición ligada, conocida como la *DIUx (2018) xView Detection Challenge*, enfocada en objetivos como reducir la resolución de imagen necesaria para la detección de objetos, mejorar la eficiencia de aprendizaje, descubrir nuevas categorías de entidades clasificables, etc. Vemos un ejemplo en la Figura 19.

[13] Página web oficial del dataset xView: http://xviewdataset.org/#dataset

Figura 19. Ejemplo de xView.
Fuente: Brian (2018)

Podemos ver, como, según se ha comentado, se trata de imágenes tomadas desde el aire o el espacio, etiquetadas con ciertos objetos ya conocidos, como edificios, aviones, vehículos terrestres, etc.

Google Open Images [14]

Considerado uno de los más extensos entre los datasets de propósito general para visión artificial, contiene aproximadamente 9 millones de imágenes, etiquetadas de diversas formas, desde anotaciones a nivel global (a qué categoría pertenece la imagen), hasta cajas con los contornos para los objetos, segmentación visual de conceptos, etc. Vemos un ejemplo en la Figura 20.

[14] Página web oficial de Google Open Images: https://opensource.google/projects/open-images-dataset

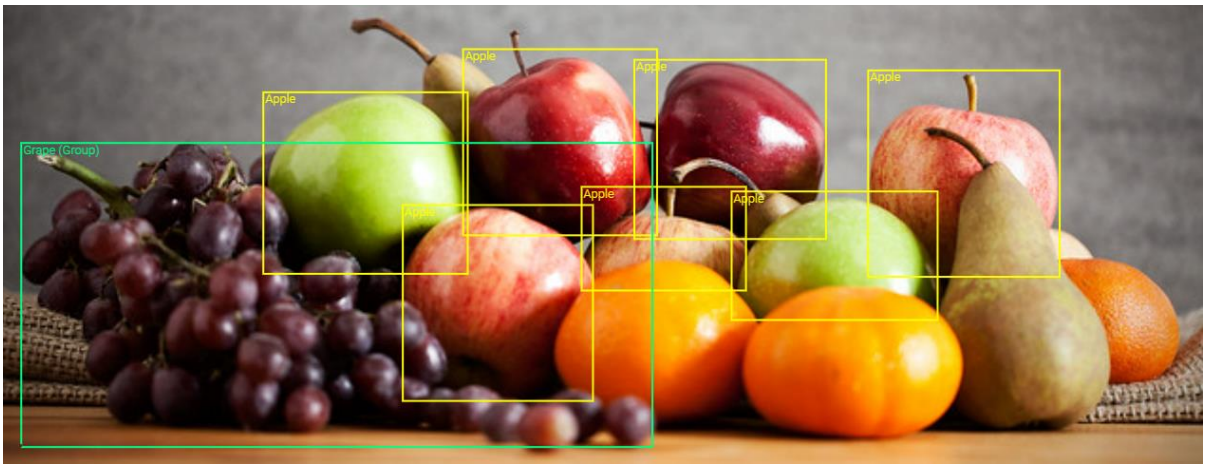

Figura 20. Ejemplos de Google Open Images.
Fuente:

En este caso, podemos apreciar un caso de detección y localización de varios objetos en la imagen, cada uno encuadrado con su correspondiente caja, señalando su posición, y descripción, señalando la categoría o clase a la que pertenece el elemento detectado.

2.6. Recapitulación

Después de este repaso conceptual, tecnológico e histórico, aunque reciente, por algunas de las aportaciones realizadas en el campo de las redes profundas aplicadas a la visión artificial, podemos llegar a una conclusión clara: las arquitecturas de CNN tienden a ser cada vez de mayor tamaño y complejidad. Aparecen año tras año nuevas estructuras y formas de organizar las capas, y menos frecuentemente, alguna nueva técnica concreta para mejorar los resultados en ciertas fases del entrenamiento, validación o test.

Se intentan realizar nuevas combinaciones sobre elementos existentes, o incluso plantear nuevos elementos conceptuales, con el fin de mejorar o sacar mayor partido a cada uno de los pasos que va dando la red en su camino hacia el resultado final, pero en el fondo, las arquitecturas de CNNs y la constitución de sus capas, así como sus tipos, no han sufrido variaciones sustanciales, que impliquen un cambio de enfoque o paradigma dentro del campo, al menos por el momento.

Además, son reseñables muchos de los problemas que esperan resolver, los cuales recapitulamos aquí:

- Tareas como la identificación y seguimiento de una persona, así como la clasificación de objetos en imágenes, o la detección de anomalías, son elementos claves en sectores como el comercio electrónico, y venta al por menor en general. Permiten, y permitirán, la creación de tiendas desatendidas, con inventariado y reemplazo de productos automáticos, así como la eliminación de aquellos defectuosos.

- Reconocimiento de escenas, y localización de objetos en imágenes, pueden tener un gran impacto en marketing y publicidad. Abrirían la posibilidad a disponer de sistemas que nos permitan identificar ubicaciones geográficas, o productos comerciales concretos, dentro de escenas de televisión o cine cotidianas, habilitando para el espectador y cliente un acceso directo a dicha información.

- Cualquier tipo de tratamiento o procesado en imágenes, como la transferencia de estilos, la colorización o restauración de imágenes o la generación de imágenes a partir de texto, puede suponer un gran salto en industrias como la audiovisual, llegando a ocupar ciertos lugares antes destinados a tareas más creativas, realizadas por humanos.

Como se ha mencionado, se trata de tareas y problemáticas de vital importancia en el mundo actual, y cada vez irán adquiriendo más peso dentro de los departamentos científico-tecnológicos de múltiples universidades y centros de investigación en todo el mundo. Al mismo tiempo, es cada vez más notorio el interés del sector privado y público en estos temas. Empresas, gobiernos y organizaciones de todo tipo pretenden aprovechar los progresos realizados por parte de la comunidad científica, especialmente en los últimos años, con el fin de desarrollar aplicaciones comerciales y obtener un beneficio económico, así como ventajas competitivas.

Es por todo esto, especialmente por la gran variedad de arquitecturas y técnicas existentes, que se vuelve una cuestión transcendental el que se realicen estudios comparativos entras ellas, y tanto investigadores como profesionales dispongan de análisis detallados del rendimiento de unas y otras, según circunstancias y casos de uso variados. De ahí nace la idea y la oportunidad del presente trabajo.

Aunque se explica con más detalle en el siguiente capítulo, por los motivos expuestos anteriormente, se concluye con el planteamiento de una comparativa entre diferentes arquitecturas de redes neuronales para visión artificial, centrando el foco en aquellas especializadas en tareas de clasificación de imágenes. Por ello, se ha realizado un análisis comparativo con el objetivo de entrenar y evaluar diferentes modelos de redes convolucionales, en diferentes escenarios de uso representados por diferentes datasets, y con la aplicación o no de algunas técnicas de IA mencionadas brevemente en apartados anteriores.

3. Objetivos y metodología de trabajo

Este capítulo sirve como enlace natural entre el estudio del dominio realizado y la contextualización de la problemática a abordar en el trabajo, presentando la aportación que se pretende conseguir con el desarrollo a realizar en este.

3.1. Objetivo general

El objetivo principal de este trabajo es:

Realizar un análisis comparativo que evalúe el rendimiento que ofrecen diferentes arquitecturas de redes neuronales centradas en tareas de clasificación de imágenes

Más concretamente, se habla aquí de redes neuronales convolucionales profundas, especializadas en diversas técnicas de visión artificial. Para ello, el estudio se basará en algunas de las arquitecturas planteadas y descritas anteriormente. Durante la última década, este tipo de redes neuronales han copado el estado del arte en cuanto a tareas relacionadas con el procesamiento de imagen, surgiendo cada año nuevas arquitecturas con novedosas aportaciones al campo.

Para lograr el objetivo mencionado, se plantea la comparativa de dos paradigmas de arquitecturas de redes neuronales: las entrenadas desde cero, y las preentrenadas en datasets distintos al utilizado en el trabajo. La idea es comprobar el resultado obtenido por cada una de ellas, al tratar de clasificar imágenes, empleando como referencia de evaluación alguno de los datasets expuestos anteriormente. Para ello, se propone recoger métricas adecuadas que permitan realizar un cotejo lo suficientemente amplio como para poder extraer conocimiento relevante acerca del desempeño de dichos modelos.

Por una parte, la intención es plantear y construir una red neuronal convolucional, a la que denominaremos *custom-cnn*, con una arquitectura sencilla pero lo suficientemente potente para clasificar imágenes con una notable precisión. Esta red se adaptará de una arquitectura existente, y luego se entrenará y evaluará desde cero. Cuenta con pocos parámetros y capas, en comparación con las grandes CNNs del capítulo anterior, y es por tanto más asumible en cuanto al tiempo de ejecución y recursos computacionales necesarios para alcanzar una precisión aceptable durante su evaluación.

Por otra parte, se utilizarán algunas de las arquitecturas de CNNs descritas previamente. No obstante, en este caso se tratará de modelos preentrenados, lo que significa que tienen un aprendizaje ya adquirido, pero sobre datasets diferentes al empleado en este trabajo. Mediante *transfer learning*, se utilizará dicho aprendizaje y se valorará el desempeño que pueden exhibir tales redes, con tan solo unas pocas adaptaciones al nuevo dataset sobre el que se quieren evaluar, pero con la opción de evitar el

realizar un nuevo proceso de entrenamiento, comprobando así si existen razones para considerar útil esa técnica de transferencia de aprendizaje.

3.2. Objetivos específicos

Con el fin de desarrollar este objetivo general, se han planteado los siguientes objetivos específicos:

1. Seleccionar los modelos, datasets y librerías más adecuadas a utilizar en el estudio, justificando los motivos de su elección frente a otras alternativas, a partir de la revisión del estado del arte realizada en el capítulo anterior.

2. Encontrar las mejores implementaciones disponibles para las arquitecturas de redes CNNs seleccionadas, y los formatos más eficientes para los datasets, así como las técnicas de preprocesado de los datos o entrenamiento del modelo óptimas para acelerar el proceso.

3. Desarrollar el análisis comparativo, ejecutando y evaluando los modelos elegidos anteriormente, recolectando para ello diferentes métricas a utilizar en el análisis posterior.

4. Discutir los resultados, tratando de interpretarlos y darles un significado, además de estudiar las ventajas y desventajas de las distintas opciones, extrayendo conclusiones relevantes.

3.3. Metodología del trabajo

Para lograr tanto el objetivo general como el listado de objetivos específicos descritos anteriormente, se ha propuesto la metodología expuesta a continuación, que consta de los siguientes pasos:

- Paso 1.

  Proceso de selección sobre un conjunto de frameworks, arquitecturas y datasets previamente descubiertos. Para ello, se realiza una búsqueda de las mejores alternativas, exponiendo los motivos que explican su relativa superioridad en ciertos escenarios planteados en este trabajo. Algunos de esos escenarios pueden ser cuestiones relacionadas con la potencia de cálculo, el tiempo de entrenamiento, o el tamaño de los modelos preentrenados o datasets cargados. Lo que se pretende es que el trabajo sea abarcable con los recursos a disposición de un solo usuario a nivel particular.

- Paso 2.

  Construcción y adaptación de las diferentes arquitecturas. En este punto se tienen en cuenta los algoritmos o modelos elegidos antes, con la idea de realizar las modificaciones necesarias para su ejecución y posterior evaluación sobre el dataset seleccionado. También se efectúan los ajustes requeridos en cuanto al preprocesamiento del dataset, las técnicas para acelerar el entrenamiento y convergencia de la red, o la visualización de resultados de cada proceso.

- Paso 3.

  Ejecución de los modelos desarrollados. Se lanza la ejecución de todos los modelos de arquitecturas seleccionadas e implementadas, con el objetivo de recolectar diferentes métricas. De esta forma, se ejecuta cada una de las arquitecturas, en sus fases de entrenamiento y evaluación (modelo *custom-cnn*), o, alternativamente, solo en sus fases de evaluación (modelos preentrenados), con el fin de reunir un compendio de medidas de diferente naturaleza, que permitirán su posterior estudio.

- Paso 4.

  Análisis de los resultados. Se interpretan las diferentes métricas recogidas en la etapa anterior, para poder sacar conclusiones relevantes que expliquen la utilidad, ventajas y desventajas de cada una de las arquitecturas, estudiando el comportamiento exhibido por cada una de ellas. Es el paso fundamental del presente trabajo, además de la propia comparativa, pues permite inferir juicios acerca de la productividad e idoneidad de las diferentes alternativas.

4. Planteamiento de la comparativa

En este capítulo se relata, de forma resumida, el proceso previo que se ha realizado para llegar desde la identificación del problema concreto a tratar, hasta la exposición de las posibles soluciones alternativas a evaluar, las cuales se detallarán aquí. De igual manera, se enumerarán y explicarán las diferentes métricas obtenidas para conformar los resultados.

Recapitulando, se ha realizado un profundo estudio del estado del arte actual en relación al aprendizaje automático, al aprendizaje profundo, y a las redes neuronales especializadas en tareas de visión artificial, conocidas como redes convolucionales. De ese análisis previo, se ha concluido que uno de los problemas predominantes actualmente en el campo de la visión artificial es el de la clasificación de imágenes, sobre el que se ha venido trabajando ampliamente a lo largo de la última década con diversos modelos de redes convolucionales, como los ya explicados AlexNet, VGGNet, ResNet, o la más reciente SENet, entre otras.

Asimismo, hemos visto como de manera paralela han ido surgiendo numerosos datasets de imágenes, especializados en el ámbito de la visión artificial, como los mencionados MNIST, COIL-100, CIFAR 10 & 100, incluso algunos nacidos no solo en el seno de universidades o grupos de investigación, sino también por parte del mundo empresarial, como el dataset Open Images de Google.

Esta proliferación en cuanto a arquitecturas disponibles y conjuntos de datos adecuados para entrenarlas, sumado a la relevancia que ostenta actualmente el campo de la visión artificial dentro de la IA, han originado el planteamiento de una comparativa entre algunos de los mencionados modelos de redes, con el fin de poder extraer conclusiones relevantes respecto al comportamiento de sus arquitecturas, y las similitudes y diferencias existentes entre ellas.

4.1. Selección de datasets

Para poder evaluar las arquitecturas seleccionadas se ha elegido uno de los datasets más conocidos y utilizados durante los últimos tiempos, en concreto, CIFAR-10.

Como se ha comentado, este dataset provee un total de 60 mil fotografías del mundo real, en forma de imágenes de 32x32 píxeles, como ejemplos para ayudar al aprendizaje de la red, estando destinadas 50 mil de ellas al entrenamiento, y las 10 mil restantes a la fase de testeo. Dichas imágenes están clasificadas hasta en 10 categorías diferentes, a saber: *airplane* (avión), *automobile* (coche), *bird* (pájaro), *cat* (gato), *deer* (ciervo), *dog* (perro), *frog* (rana), *horse* (caballo), *ship* (barco) y *truck* (camión).

Todas las categorías son mutuamente excluyentes. Sin embargo, se puede apreciar que en ocasiones guardan cierta relación entre ellas, como en la pareja coche/camión, ambos vehículos, o también en el caso de gato/perro, animales similares en cuanto a complexión física. En sendos casos, hablamos de clases distintas pero relacionadas, lo que añade complejidad al proceso de aprendizaje en el caso de querer clasificar imágenes de alguno de los mencionados grupos, pues su semejanza hará que sea más difícil discernir, por ejemplo, entre un gato o un perro, que entre un pájaro y un coche.

Los criterios para la elección del citado dataset han sido varios.

- En primer lugar, está enfocado en la clasificación de imágenes, aspecto central de atención en el presente trabajo, y, por lo tanto, se antoja especialmente apropiado, dentro de aquellos encuadrados en tareas de visión artificial. Además, se trata de imágenes reales, no aisladas, sino integradas en un entorno, lo que añade dificultad a su clasificación.

- Por otro lado, es un dataset que se puede considerar sencillo, en relación a otros, como el propio CIFAR-100 o el colosal Google Open Images, con cerca de 9 millones de ejemplos, que excederían las posibilidades de un usuario particular, superando por mucho el alcance pretendido en el presente trabajo. A su vez, es mucho más complejo que otros como MNIST o Fashion-MNIST, los cuales contarían con el riesgo de quedarse demasiado cortos a la hora de obtener conclusiones destacadas, por su relativa sencillez. Es por ello que se considera como un término medio adecuado, en cuanto a su tamaño, así como en referencia a la variedad de imágenes que ofrece, y, por tanto, suficiente para poner en apuros a las diferentes arquitecturas y poder comparar los resultados obtenidos por las mismas.

- Asimismo, el hecho de que cuente con un número de ejemplos reducido, lo hace cómodo y manejable para un usuario particular. Esta facilidad de operabilidad se manifiesta en el tamaño en memoria o disco que puede llegar a ocupar, siendo de unos 163 MB en su última versión para Python. Consecuencia directa de ello es la menor capacidad de cómputo requerida para su entrenamiento, permitiendo acortar dicho proceso y obtener muy buenos resultados en menos de 1 hora, especialmente en el caso de usar plataformas en la nube como Google Colaboratory[15].

- Por último, como se detallará más adelante, se trata de un dataset distinto a *ImageNet*, del cual se valieron las redes preentrenadas utilizadas en la comparativa. De esta forma, tenemos un nuevo conjunto de datos completamente diferente, y más sencillo en este caso, sobre el que evaluar la efectividad de aplicar la transferencia de conocimiento para dichos modelos, lo que permitirá a su vez inferir algunas conjeturas acerca de la utilidad de la mencionada técnica.

[15] Página web de Google Colaboratory: https://colab.research.google.com/

4.2. Selección de arquitecturas

Como ya se ha planteado, el objetivo de la comparativa ha sido enfrentar, principalmente, dos grupos de arquitecturas. Por un lado, tenemos un modelo de red neuronal adaptado desde una arquitectura existente, al que llamamos *custom-cnn*. Por otro, tenemos dos modelos de redes preentrenadas en ImageNet, dataset diferente a CIFAR-10, el utilizado ahora en la comparativa para evaluarlas.

De la misma forma que ocurría con la elección del dataset, el hecho de utilizar redes pequeñas, o incluso preentrenadas, corresponde a la necesidad de reducir los requerimientos de cómputo, espacio y tiempo que podrían significar redes mayores, una vez más, con el propósito de contener el ámbito del estudio.

4.2.1. Modelo *Custom CNN*

Para el caso de este modelo *custom-cnn*, se ha optado por una arquitectura extraída de Kumar (2018/2020), y adaptada para los propósitos del estudio. Su elección se ha basado en los buenos resultados que demuestra para el dataset objetivo del estudio, rozando el 90% de precisión no solo durante el entrenamiento y validación, sino a lo largo de la fase de test, como se verá más adelante.

También se ha considerado dicho modelo como una buena opción, dentro de las disponibles, por la implementación que realiza de varias técnicas que han demostrado ser bastante útiles en términos de mejora del aprendizaje de una red, permitiendo no solo una mayor velocidad en su ejecución, sino también mejores resultados y mayor robustez del modelo entrenado ante nuevos ejemplos.

Cuenta con tan solo seis capas convolucionales, además de algunas intermedias para optimizar y acelerar el aprendizaje, lo que la convierte en una estructura relativamente sencilla, e incluso pequeña, en comparación a los grandes modelos analizados antes. Tiene un total de 309.290 parámetros, 308.394 de los cuales son entrenables y 896 no entrenables. Se muestra su estructura en la Figura 21.

Como se puede apreciar, está conformada por la repetición de **seis bloques principales**, cada uno de los cuales cuenta con una capa convolucional (*Conv2D*) seguida de su función de activación ELU (*Activation*) y una capa de normalización (*BatchNormalization*). Además, podemos encontrar **tres bloques secundarios**, formados por una capa max pooling (*MaxPooling2D*) y una dropout (con valores del 20%, 30% y 40%, respectivamente para cada bloque), y situados después de los bloques principales segundo, cuarto y sexto. La arquitectura finaliza con una capa (*Flatten*), la cual permite aplanar los valores que recibe, y sirve de conexión entre el último bloque secundario y la capa de salida densa de 10 neuronas (10 categorías del dataset), con activación *softmax*, idónea para ser usada en problemas de clasificación multiclase, como el que nos ocupa.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
activation (Activation)	(None, 32, 32, 32)	0
batch_normalization (BatchNo	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
activation_1 (Activation)	(None, 32, 32, 32)	0
batch_normalization_1 (Batch	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496
activation_2 (Activation)	(None, 16, 16, 64)	0
batch_normalization_2 (Batch	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36928
activation_3 (Activation)	(None, 16, 16, 64)	0
batch_normalization_3 (Batch	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73856
activation_4 (Activation)	(None, 8, 8, 128)	0
batch_normalization_4 (Batch	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584
activation_5 (Activation)	(None, 8, 8, 128)	0
batch_normalization_5 (Batch	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 10)	20490

Figura 21. Arquitectura de la red convolucional custom-cnn.
Fuente: elaboración propia

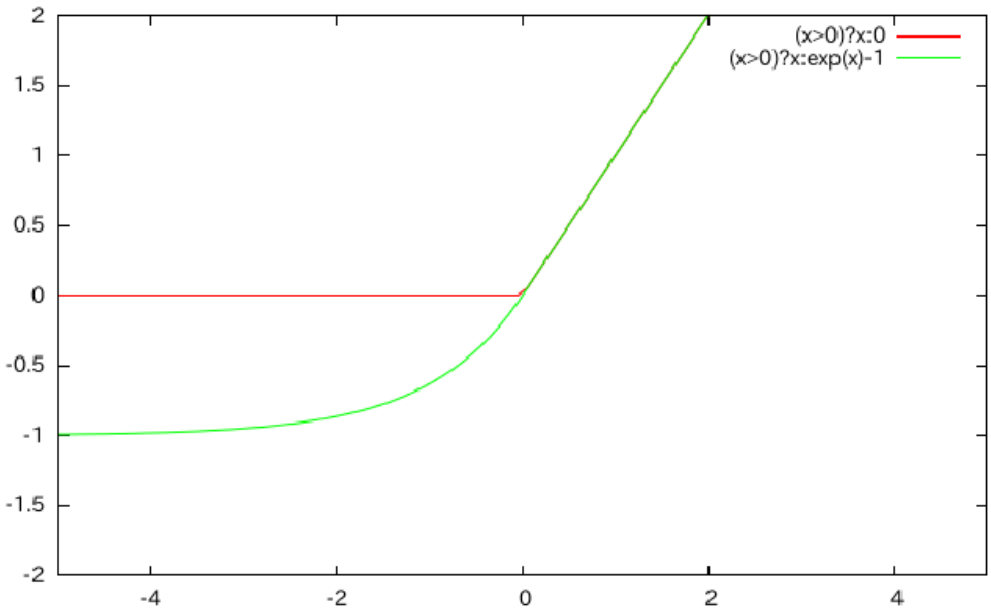

Figura 22. Funciones de activación ReLU (rojo) y ELU (verde).
Fuente: Kurita (2017)

Se hace útil aquí destacar su función de activación ELU, muy similar a la ampliamente empleada ReLU, y cuyas similitudes se muestran en la Figura 22. En ella se puede apreciar cómo, mientras que ambas son idénticas para valores positivos, difieren para los negativos.

Dicha capacidad de producir salidas negativas, que no tiene ReLU, es una de las ventajas respecto a esta última. Otra sería la de evitar algunos inconvenientes conocidos, como el *dying ReLU problem* (Leung, 2021), según el cual, con un gradiente suficientemente grande, podría ocurrir que los pesos de la neurona cambiasen de tal forma que esta nunca volviera a activarse, manteniéndose siempre a cero.

De igual modo, este tipo de activación también presenta algunas desventajas, como mayor carga de cálculo, al añadir una operación exponencial, además de que su éxito depende parcialmente de un parámetro α, el cual no es aprendido por la red, sino que debe ser ajustado manualmente.

En el siguiente capítulo, destinado al desarrollo de la comparativa, se describirán de manera minuciosa algunas de las particularidades propias de la ejecución de esta red durante su fase de entrenamiento, y no tanto de la estructura de la misma.

4.2.2. Redes preentrenadas

En lo referente a los modelos preentrenados, se trata de dos arquitecturas preentrenadas en *ImageNet*, y disponibles mediante el paquete *applications* que ofrece la librería Keras integrada en el ecosistema Tensorflow 2, en concreto en su versión 2.6.0. Asimismo, como se verá más adelante, Keras proporciona una serie de utilidades para aplicar la técnica de *transfer learning*, lo cual sumado a su amplio repertorio de modelos preentrenados, le convierte en una librería adecuada para esta labor.

Antes de entrar en detalle en cada una, se brinda una tabla comparativa *a priori* de las implementaciones elegidas para los modelos seleccionados, en este caso *vgg16* y *resnet50*, correspondientes a las arquitecturas tradicionales VGGNet y ResNet, explicadas previamente.

Model	Parameters	Size (MB)	Top-1 Accuracy	Top-5 Accuracy
VGG16	138,357,544	528	0.713	0.901
ResNet50	25,636,712	98	0.749	0.921

Figura 23. Tabla comparativa de las arquitecturas VGGNet 16 y ResNet 50.
Fuente: Team (2021)

VGG16 (VGGNET 16)

Como se puede ver en la Figura 23, el modelo original de *vgg16* cuenta con un total de 138 millones de parámetros, y un tamaño de más de 500 MB, haciendo notar su mayor envergadura respecto al otro modelo preentrenado propuesto, ResNet 50. Por otro lado, también exhibe un rendimiento ligeramente inferior al ofrecido por su contraparte, pues tiene peores marcas de clasificación en *ImageNet*, su dataset original, con un 71.3% de *Top-1 Accuracy* y un 90.1% de *Top-5 Accuracy*.

Se recuerda aquí la implementación de 16 capas del modelo general VGGNet explicado en capítulos anteriores. De forma resumida, su arquitectura tradicional se basa en capas convolucionales (**C**), capas max. pooling (**P**) y capas densas (**D**), dispuestas formando el siguiente esquema. Así, obviando las capas max. pooling, nos da un total de 16 capas, 13 de ellas convolucionales y 3 finales densas, formando la parte de la red completamente conectada, encargada de componer la salida.

CCP – CCP – CCCP – CCCP – CCCP – DDD

Esta arquitectura inicial ha sufrido una serie de modificaciones, siguiendo el guion propuesto por Paul (2018/2021) en su repositorio GitHub, con el fin de adaptarla al ejercicio que nos ocupa, llegando a la estructura mostrada en la Figura 24.

Conserva solo los 3 primeros bloques del modelo original, incluida la última capa de su tercer bloque (*block3_pool*), a partir de la cual se ha implementado el resto de la estructura. Se comienza añadiendo una capa de pooling (*GlobalAveragePooling*), así como una de normalización (*BatchNormalization*), a objeto de regularizar la salida del bloque reutilizado. Por último, su salida se estructura en dos capas densas (*Dense*), de 256 neuronas cada una, seguidas de una capa dropout y una última capa densa para habilitar la salida de los resultados, de 10 neuronas, con activación *softmax*.

De esta manera, se han logrado reducir los más de 138 millones de parámetros iniciales del modelo base, a tan solo 1.8 millones que conforman la arquitectura combinada. Además, de ese total, 134 mil de ellos son entrenables, correspondientes a las nuevas capas añadidas. Por otro lado, los restantes

1.7 millones de parámetros no entrenables se refieren al conjunto de capas reutilizadas (y congeladas), las cuales reaprovechan el preentrenamiento realizado antes.

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 48, 48, 3)]	0
block1_conv1 (Conv2D)	(None, 48, 48, 64)	1792
block1_conv2 (Conv2D)	(None, 48, 48, 64)	36928
block1_pool (MaxPooling2D)	(None, 24, 24, 64)	0
block2_conv1 (Conv2D)	(None, 24, 24, 128)	73856
block2_conv2 (Conv2D)	(None, 24, 24, 128)	147584
block2_pool (MaxPooling2D)	(None, 12, 12, 128)	0
block3_conv1 (Conv2D)	(None, 12, 12, 256)	295168
block3_conv2 (Conv2D)	(None, 12, 12, 256)	590080
block3_conv3 (Conv2D)	(None, 12, 12, 256)	590080
block3_pool (MaxPooling2D)	(None, 6, 6, 256)	0
global_average_pooling2d_1 (	(None, 256)	0
batch_normalization_1 (Batch	(None, 256)	1024
dense_3 (Dense)	(None, 256)	65792
dense_4 (Dense)	(None, 256)	65792
dropout_1 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 10)	2570

Figura 24. Arquitectura de la red convolucional VGGNet 16 adaptada.
Fuente: elaboración propia

RESNET50 (RESNET 50)

A raíz de lo expuesto en la Figura 23, es destacable el hecho de que, originalmente, esta arquitectura *resnet50* ofrecía una solución mucho más liviana (~100 MB) que la recién comentada VGGNet, con apenas una quinta parte del tamaño del modelo de esta, así como unas cinco veces menor cantidad de parámetros (25 millones). A pesar de ello, se puede observar cómo *resnet50* obtiene mejores resultados en el *Top-1* y *Top-5 Accuracy* que su contrapartida *vgg16*.

Al igual que se ha hecho con la arquitectura anterior, este modelo inicial también ha tenido una serie de variaciones respecto al original, las cuales se han extraído de Dabydeen (2019), y pueden verse reflejadas en la Figura 25, que se muestra a continuación.

Layer (type)	Output Shape	Param #
up_sampling2d_6 (UpSampling2	(None, 64, 64, 3)	0
up_sampling2d_7 (UpSampling2	(None, 128, 128, 3)	0
up_sampling2d_8 (UpSampling2	(None, 256, 256, 3)	0
resnet50 (Functional)	(None, 8, 8, 2048)	23587712
flatten_2 (Flatten)	(None, 131072)	0
batch_normalization_6 (Batch	(None, 131072)	524288
dense_6 (Dense)	(None, 128)	16777344
dropout_4 (Dropout)	(None, 128)	0
batch_normalization_7 (Batch	(None, 128)	512
dense_7 (Dense)	(None, 64)	8256
dropout_5 (Dropout)	(None, 64)	0
batch_normalization_8 (Batch	(None, 64)	256
dense_8 (Dense)	(None, 10)	650

Figura 25. Arquitectura de la red convolucional ResNet 50 adaptada.
Fuente: elaboración propia

En primer lugar, antes de que la entrada llegue al modelo base *resnet50* (*Functional*), se introducen 3 capas de reescalado (*UpSampling2D*) con factores duplicativos para las filas y columnas de cada imagen, con el objetivo de aumentar el tamaño de entrada en cada capa, pasando así de los 32x32x3 píxeles que admite CIFAR-10, hasta llegar a los 256x256x3 píxeles requeridos por ResNet como input, tamaño correspondiente al original de las muestras en *ImageNet*. Una alternativa a la introducción de estas capas hubiera sido la de realizar esta transformación durante el preprocesamiento de los datos.

Seguidamente, el flujo de datos discurre a través del modelo base correspondiente a la arquitectura ResNet, y se aplana su salida (*Flatten*), desembocando en sendas capas completamente conectadas (*Dense*) de 128 y 64 neuronas, respectivamente. Estas capas densas cuentan a su entrada con una capa extra de normalización (*Batch Normalization*), así como a su salida con una capa dropout, que aplica el 50% de inactividad para las neuronas de cada capa densa.

Al final, se cuenta con la tradicional capa de salida, de 10 neuronas completamente conectadas, relativas a las 10 categorías de clasificación del dataset CIFAR-10. Una vez más, cuenta con una activación de tipo *softmax*, cuyas probabilidades combinadas aúnan el 100%.

Finalmente, destacar que no se ha congelado ninguna de las capas del modelo original, sino que se parte de los pesos adquiridos para *ImageNet*, y se reentrena la red desde ahí, permitiendo que tanto las capas del modelo base como las nuevas capas añadidas modifiquen los valores de sus parámetros de entrenamiento en cada iteración o época. De esta manera, partiendo de los 23 millones de parámetros del modelo base, alcanzamos un total de 40 millones de parámetros entrenables, correspondiendo los últimos 17 millones a las dos capas densas finales.

Cabe reseñar la pequeña discrepancia entre los valores pertenecientes al modelo original de *resnet50*, con más de 25 millones de parámetros, respecto a los 23 millones y medio con los que cuenta la implementación utilizada aquí. Dicha diferencia responde a que, a la hora de utilizar la arquitectura original, se ha eliminado su última capa densa, de 1000 neuronas con activación *softmax*, encargada de la salida para el dataset original, *ImageNet*, el cual cuenta con esas 1000 categorías de clasificación.

4.3. Métricas utilizadas

Por último, es momento de explicar aquí las distintas métricas recolectadas como consecuencia de las diversas ejecuciones de los modelos, las cuales ofrecen a su vez un camino para, en un paso posterior, inferir posibles significados que manifiesten las similitudes, diferencias y particularidades acerca de las arquitecturas utilizadas, así como de sus parámetros de configuración. Nos centraremos en:

- ***Accuracy***.

  Como se recoge en Universidad Internacional de La Rioja (2019), esta métrica, una de las más simples y a su vez más utilizadas que existen, se conoce también como ratio de éxito, y expresa el porcentaje de predicciones correctas que realiza la red neuronal sobre el conjunto total de predicciones durante su fase de entrenamiento. Poniendo un ejemplo, si la red hace un total de 100 predicciones, y 75 de ellas son correctas, es decir, iguales al valor real (*ground truth*), se dirá que dicha red tiene una *accuracy* de 75 sobre 100, o lo que es lo mismo, del 75%. Para distinguirla, se puede hacer referencia a ella también como *training accuracy*, dado que como veremos ahora, esta métrica se puede aplicar igualmente en la validación y el test.

- ***Loss***.

  En primer lugar, como bien se indica en Minaee (2019), conviene aclarar que la pérdida (*loss*) de la red no se trata de una métrica *per se*. Esta medida se calcula como resultado de una función de pérdida, la cual es la encargada de calcular el rendimiento que está teniendo el modelo

durante su fase de entrenamiento, paso por paso, época por época, y con ello ir ajustando la red para mejorar su aprendizaje. Es decir, expresa cuán cerca o lejos del resultado real, en forma de margen de error, se encuentra la red en cada predicción.

Las métricas se caracterizan por ser herramientas útiles para observar el desempeño de una red *a posteriori* de su ejecución, y, además, no tienen por qué ser matemáticamente diferenciables (derivables). No obstante, en el caso de la pérdida, sí que debe ser una función derivable, pues de ello depende el proceso de *backpropagation* que ejecuta el modelo, el cual utiliza precisamente estas derivadas para calcular los nuevos gradientes de la función, que marcan la pauta para actualizar algunos parámetros del entrenamiento, como los pesos.

Puede darse el caso de que una medida empleada comúnmente como métrica sea además derivable, permitiendo que sea usada tanto para la función de coste de la red, como para la recolección de métricas una vez finalizada la ejecución. Es el caso de la métrica MSE (*Mean Square Error*), utilizada frecuentemente como función de coste en problemas de regresión.

- ***Validation accuracy / Test accuracy***.

  Se rigen por la misma fórmula que la recientemente explicada *accuracy* (o *training accuracy*), pero se aplican sobre conjuntos de datos diferentes a los de entrenamiento.

  En el caso de la *validation accuracy*, se calcula igual que la *training accuracy*, esto es, en cada época, pero sobre una fracción del conjunto de datos de entrenamiento, fracción conocida como conjunto de datos de validación. Permite ir evaluando el desempeño de la red, paso a paso, en un conjunto de datos diferente al que se está usando para entrenarla.

  A su vez, esta métrica habilita la introducción de nuevas técnicas, como la llamada *early stopping*, según la cual, si la *validation accuracy* no está mejorando de manera significativa tras cierto número de épocas o tras cierto periodo de tiempo, se convierte en un claro indicativo de que se recomienda detener el entrenamiento, pues no es probable que mejore aun dando más tiempo a la red para que entrene. De esta forma se ahorran recursos computacionales, y tiempo.

  Aunque la fase de validación es opcional, y no siempre se realiza, sí que ha sido utilizada en el presente trabajo, y por ello se ha explicado aquí la métrica correspondiente.

  Si hablamos de la *test accuracy*, es calculada una sola vez, cuando finaliza el proceso de entrenamiento, y se realiza sobre un conjunto de datos diferente al de entrenamiento o validación, conocido como conjunto de test. Permite conocer el comportamiento final de la red, una vez entrenada, ante datos distintos a los que ha visto previamente.

En relación a la *accuracy*, si, por ejemplo, la *validation accuracy* fuese superior a la *training accuracy*, y se observase una amplia diferencia entre ambas, existirían elevadas probabilidades de que la red hubiera caído en *overfitting*, aprendiendo demasiado bien los datos de entrenamiento, casi memorizándolos, y por tanto perdiendo capacidad de generalización sobre ejemplos no vistos anteriormente, haciendo que, con casi total seguridad, sus resultados para la *test accuracy* no fuesen lo suficientemente satisfactorios.

- ***Validation loss / Test loss***.

  Igual que en el caso anterior, tenemos medidas para la pérdida que experimenta la función, no solo a lo largo de la fase de entrenamiento, sino también durante la validación que acompaña cada paso de dicho entrenamiento. Esta métrica se conoce como *validation loss*.

  Por otro lado, de la misma forma que tenemos una métrica única y final, post-entrenamiento, conocida como *test accuracy*, encontramos el equivalente cuando hablamos de *test loss*.

  Generalmente, para los tres tipos de *accuracy*, ésta será inversamente proporcional a la *loss* calculada, pues a mayor *accuracy*, deberíamos percibir valores menores para la pérdida, y a la inversa, si tuviéramos una medida de *loss* excesivamente grande, en su contexto, querría decir que la *accuracy* alcanzada sería escasa.

- ***Tiempo de ejecución y número de épocas***.

  Es el tiempo total que tarda la red durante el proceso de entrenamiento, expresado en horas, minutos y/o segundos. Es directamente proporcional al número de épocas que ejecuta la red, pues a más épocas, más tiempo de ejecución, y viceversa.

  Sumando la duración de cada época, tendríamos el tiempo total exacto que habría necesitado la red para entrenarse. Sin embargo, generalmente cada época suele tardar el mismo tiempo en promedio, por lo que, multiplicando ese tiempo por el número total de épocas, también podríamos obtener el resultado de tiempo total, aunque en este caso de forma aproximada.

- ***Consumo de memoria***.

  Es una medida bastante simple, que sencillamente expresa la cantidad de RAM (memoria principal) que ocupa tanto el modelo de la red, con los valores de sus parámetros, como el dataset utilizado. En caso de que el dataset sea excesivamente grande, se puede ir leyendo desde disco (memoria secundaria), lo que reduciría el requisito de consumir una mayor cantidad de RAM, pero ralentizaría el proceso de aprendizaje, pues se tendría que estar continuamente realizando lecturas de disco, el cual es un tipo de memoria con tiempos de acceso mucho más lentos que la memoria principal.

Por ese motivo, generalmente, y siempre que los recursos disponibles lo permitan, es recomendable cachear (cargar en memoria desde disco) el dataset, preferiblemente de forma completa, eliminando esas consultas a la memoria secundaria, y reduciendo por tanto los tiempos empleados por la red para su aprendizaje.

El consumo de memoria principal puede ser algo a considerar no solo en el caso del dataset, sino también para el propio tamaño del modelo. Caso conocido es por ejemplo el de la ya comentada AlexNet, la cual, debido a las limitaciones en la memoria de las tarjetas gráficas de la época, tuvo que ser dividida, poniendo la mitad del modelo en una GPU, y la otra mitad en otra GPU diferente, dado que el tamaño era superior al disponible para la memoria de video. De esta forma, se entrenó de forma paralela y simultánea sobre dos componentes distintos.

- ***Espacio en disco***.

  Al igual que el consumo de memoria principal ejemplifica el tamaño que ocupan en memoria RAM los recursos que emplean la red para su aprendizaje (modelo, dataset, etc.), el espacio en disco es una métrica que nos habla del espacio ocupado por los mismos, pero en este caso en la memoria secundaria del ordenador.

  Como es razonable, dichos recursos también pueden estar repartidos entre la memoria principal y la secundaria, aunque lo ideal, como ya se ha abordado, es que todos estén disponibles en el tipo de memoria con un acceso más rápido, es decir, la memoria principal, o memoria RAM. De esta forma, el proceso estará mucho más optimizado desde un punto de vista meramente de hardware, y ya solo por eso, su ejecución en cada una de las fases será más veloz.

5. Desarrollo de la comparativa

En este capítulo se detalla la implementación progresiva de los algoritmos correspondientes a los diferentes modelos, realizada en el lenguaje de programación Python[16]. Además, se ha hecho uso del ecosistema Tensorflow en su versión 2, el cual trae integrada la librería Keras. Como ya se ha hecho mención, de todas las variantes de Tensorflow disponibles se ha usado aquella que utiliza una API en Python como medio de interacción, aunque existen otras disponibles como C++, Java, JavaScript, etc.

Es bien sabido que Python no se caracteriza por ser un lenguaje especialmente rápido en su ejecución, debido sobre todo al hecho de que es interpretado. Sin embargo, gran parte del núcleo principal de Tensorflow para Python está implementado usando una combinación de otros dos lenguajes. Por un lado, se utiliza una versión altamente optimizada de C++, a muy bajo nivel, logrando con ello una mayor velocidad en la ejecución de cada sentencia. Por otro, se utiliza CUDA, creado por la empresa Nvidia, un lenguaje de programación especializado en la ejecución sobre GPUs.

En el caso del modelo *custom-cnn*, se ha tenido que adaptar la solución partiendo de la arquitectura explicada previamente, desde de la construcción por capas de modelo, hasta su compilación, entrenamiento, y posterior evaluación durante su fase de test.

Para las redes preentrenadas, dado que se trata de redes, como su propio nombre indica, entrenadas anteriormente, ya cuentan con valores para los parámetros de entrenamiento, adquiridos en un proceso anterior sobre *ImageNet*. A pesar de ello, también se realiza una fase de entrenamiento sobre ellas, junto con la correspondiente fase posterior de evaluación.

5.1. Puntualización acerca de las tecnologías empleadas

A pesar de estar lejos de los más grandes que existen, dada la magnitud del dataset utilizado y el número total de imágenes a procesar, se ha hecho uso de Google Colaboratory (abreviando, Colab), con el fin de aprovechar los recursos computacionales que Google ofrece a través de su plataforma. Dispone tanto de una versión gratuita (Free) como otra de pago (Pro).

Para la solución propuesta se ha utilizado la versión Pro, lo cual se indica explícitamente aquí, pues existen diferencias en cuanto a los recursos que brinda, ofreciendo esta versión mayor cantidad de RAM, espacio en disco, así como modelos de GPU más recientes y rápidos. Esto permitirá acortar de forma notable los tiempos de entrenamiento, especialmente en el caso de la red sin preentrenamiento, pues se hicieron pruebas con la versión gratuita, e incluso con el hardware doméstico particular disponible, y los tiempos de entrenamiento de dicha red se hacían excesivamente largos, aumentando de manera

[16] Página web oficial de Python: https://www.python.org

notable el tiempo medio por época. En una situación como esta, en la que varias ejecuciones del mismo modelo eran requeridas para obtener una buena comparativa, esto se volvía una desventaja importante.

De hecho, según Radečić (2020), a fecha de 2020, en los modelos de GPU ofrecidos por Google se podía observar diferencias de hasta el doble de rendimiento bruto ofrecido según qué versión de Colab se estuviera utilizando, con modelos de GPU Nvidia K80 para la versión Free y Nvidia P100 para la versión Pro. Con una potencia bruta aproximada de 4 TFLOPS (*Tesla K80 Specs*, 2014) para la primera opción, mientras que la segunda doblaba holgadamente esos números, con hasta 9.5 TFLOPS (*Tesla P100 Specs*, 2016), ambas medidas en precisión simple (FP32 – Floating Point 32).

5.2. Modelo *Custom CNN*

Teniendo en cuenta lo mencionado sobre la justificación del uso de este modelo , así como lo tocante a su estructura estática, es decir, su arquitectura, pasamos ahora a la descripción de su comportamiento dinámico, durante su ejecución y entrenamiento, para a continuación, mostrar los resultados obtenidos por la red implementada con dicha arquitectura.

Para esta red neural denominada internamente *custom-cnn* se ha utilizado la arquitectura de capas descrita en el anterior capítulo, y además de ello, se han seguido una serie de ideas que han conducido a la aplicación de varias técnicas y métodos para mejorar el proceso, tanto desde el preprocesamiento de los datos, como durante el entrenamiento y convergencia de la red.

5.2.1. Entrenamiento

Primeramente, cabe destacar las tareas de preprocesamiento de los datos realizadas en la implementación de esta red para el dataset objetivo. Por un lado, se comienza cargando el dataset *cifar10* y dividiendo sus ejemplos tal y como vienen por defecto, en un conjunto para entrenamiento (50 mil) y otro para validación (10 mil), el cual será utilizado también para test.

A cada uno de ellos se le aplica una normalización de unidad tipificada, también conocida como *z-score*, que consiste en aplicar dos operaciones consecutivas a cada dato, o píxel de la imagen en este caso. Siguiendo a Stackoverflow (2013), se empieza restando el valor del promedio de la población o muestra, con el objetivo de centrar los datos en torno a un eje, en este caso bien podría ser 0 (negativo: debajo del valor promedio, positivo: encima), consiguiendo con ello modelos mucho más estables y robustos desde el punto de vista matemático.

A continuación, se divide el resultado obtenido entre el valor de la desviación típica para la misma población o muestra, logrando así estandarizar los datos, con la ventaja de alcanzar valores equitativos

para los pesos en todos los canales de entrada. Esto último podría no ser del todo adecuado para enfoques distintos al aquí planteado, por lo que es una medida a tratar con precaución.

Posteriormente, se codifican los vectores correspondientes a las etiquetas de entrenamiento y test en un formato conocido como *one-hot*, caracterizado porque las únicas combinaciones de valores que admite son aquellas para las cuales solo uno de ellos tiene un valor alto (1), mientras que el resto tienen un valor bajo (0). Este tipo de codificación cuenta con varias ventajas, especialmente en el ámbito del acceso y la modificación de los datos, siendo operaciones computacionalmente rápidas.

Después de eso, se aplica la técnica *data augmentation*, explicada anteriormente, con el fin de ampliar el escueto conjunto de datos de entrenamiento de partida, de tan solo 50 mil ejemplos, añadiendo variaciones de forma, posición o tamaño a las imágenes de los ejemplos, enriqueciendo así la información de la que dispone la red para su entrenamiento.

Finalmente, se compila y entrena el modelo construido, con el conjunto de datos de entrenamiento ya preprocesado y aumentado, aplicando un tamaño de batch de 64 elementos, lo que nos deja con 781 pasos por época (*steps/epoch*). También se proporciona al proceso de entrenamiento un conjunto de datos para validación, siendo este, como se ha comentado, el mismo que el utilizado más tarde para su evaluación o test. El optimizador para el entrenamiento es *RMSprop*, y la función de coste o pérdida empleada es la conocida como *categorical crossentropy*.

Como apunte adicional, se explica aquí el mecanismo empleado para ajustar el valor de la tasa de aprendizaje de la red, según el instante de ejecución en el que se encuentra en cada momento. La lógica implementada es sencilla: con un valor base de 0.001 para las épocas #1 a 75#, un valor reducido a la mitad (0.0005) desde la época #76 a la #100, y una última reducción al 0.0003 a partir de la #101.

De esta forma, se aplica una técnica conocida como ***adaptive learning rate***, mediante un planificador de la tasa de aprendizaje (*learning rate scheduler*), con el objetivo de reducir el impacto de la misma según se aproxima el final de dicho proceso. Mientras que inicialmente se tiene una tasa más alta, permitiendo aumentar la velocidad de convergencia de la red en esos primeros momentos de mayor incertidumbre, cuando se acercan las últimas etapas del proceso, dicha tasa se ve reducida de modo progresivo, con la intención de que la red aprenda más lentamente, y con ello, de forma generalmente más fiable, posibilitando afinar sus resultados con mayor nivel de detalle.

5.2.2. Resultados

Llegados a este punto, es el momento de mostrar los resultados obtenidos. El modelo correspondiente a la arquitectura de red *custom-cnn*, con los parámetros de entrenamiento mencionados antes, y siempre hablando de su ejecución sobre el dataset *cifar10*, ha conseguido los resultados mostrados en la Figura 26, donde aparece el valor de cada métrica según la época de ejecución alcanzada.

Métrica / Época	#60	#100	#150	#300
Tiempo	24m 19s	39m 5s	58m 45s	2h 3m 4s
Accuracy	0.8498	0.8823	0.8877	0.8967
Loss	0.5905	0.4524	0.4342	0.4019
Val. / Test Accuracy	0.8575	0.8701	0.8871	0.8768
Val. / Test Loss	0.5876	0.5188	0.4707	0.4909

Figura 26. Resultados del entrenamiento de la arquitectura custom-cnn*.*
Fuente: elaboración propia

Como podemos observar, siendo de esperar, los valores de *accuracy* más altos durante el entrenamiento se registran cuando la red ejecuta el mayor número de épocas. Se alcanza el mejor valor, de un 89.67% de *accuracy*, cuando se llega a las 300 épocas de entrenamiento, tras poco más de 2 horas de ejecución. Lógicamente, los valores para la pérdida (*loss*) son inversamente proporcionales.

En este caso, no se presenta de forma aislada el valor para la *test accuracy* o *test loss*, pues en todas las ejecuciones se usó el mismo conjunto de datos tanto para la validación como para la evaluación o test, coincidiendo así, por tanto, los valores de validación del modelo en su última época, con los valores de test obtenidos después del entrenamiento de la red.

Respecto al consumo de memoria o espacio ocupado por el dataset CIFAR 10, como se comentó previamente, y se repite aquí, siendo válido para las tres arquitecturas, dicho dataset tiene un tamaño aproximado de tan solo 150 MB, siendo por lo tanto un impacto despreciable en la necesidad de recursos computacionales para poder procesarlo.

Por último, en relación al consumo de memoria RAM o de espacio en disco, tanto para el modelo como para sus pesos, nos encontramos con menos de 2 MB de tamaño total (~1.23 MB). Sin embargo, si se comprobó durante el entrenamiento, que el espacio utilizado por el proceso correspondiente a la memoria de video, llegaba fácilmente a los 10 GB / 16 GB totales que tiene la GPU, maximizando y aprovechando el paralelismo intrínseco de estos procesadores.

5.3. Redes preentrenadas

De forma similar a como se hizo para la arquitectura *custom-cnn*, es menester hablar aquí acerca de los pormenores relativos al entrenamiento y ejecución de los modelos preentrenados *vgg16* y *resnet50*. En el caso de estas redes, tenemos la particularidad de que en ambas se utiliza la técnica conocida como *transfer learning*, pero aplicada de forma muy distinta para una y otra.

Mientras que una de las redes (*vgg16*) reutiliza solo parte de la arquitectura original, descartando algunas de sus capas y congelando los pesos de las restantes, la otra (*resnet50)*, reutiliza la arquitectura completa, y no solo eso, sino que permite que las capas reutilizadas se vuelvan a entrenar sobre el nuevo dataset *cifar10*, junto con el resto de nuevas capas añadidas.

5.3.1. Entrenamiento

VGGNET 16

Se comienza hablando, como antes, de las tareas de preprocesamiento aplicadas a los datos. Inicialmente, se carga en memoria el dataset *cifar10* y se dividen los 60 mil ejemplos en los dos conjuntos predefinidos: entrenamiento (50 mil) y test (10 mil). Más tarde, el conjunto de entrenamiento divide sus 50 mil ejemplos, asignando el 85% para el nuevo conjunto de entrenamiento resultante, y reservando el 15% para un conjunto de validación a utilizar durante el aprendizaje. Así, tenemos finalmente un total de tres conjuntos de imágenes totalmente excluyentes: entrenamiento, validación y test.

Una vez más, los vectores correspondientes a las etiquetas de las categorías de salida (Y) se codifican en el formato *one-hot*, para los tres grupos de etiquetas. Con los conjuntos correspondientes a los ejemplos de entrada (X), se aplica una redimensión de cada imagen, pasando de los 32x32x3 a los 48x48x3 píxeles que admite la arquitectura *vgg16* como tamaño mínimo.

De nuevo, se vuelve a aplicar *data augmentation*, pero, mientras que para el caso de *custom-cnn*, tal técnica solo se aplicaba sobre el dataset de entrenamiento, en esta arquitectura se aplica tanto al grupo de entrenamiento como al de validación, con generadores individuales de imágenes aleatorias para cada conjunto de datos, pero ambos realizando los mismos tipos de transformaciones.

Por último, se compila el modelo utilizando *Adam* con un *learning rate* de 0.0004 como optimizador, y se asigna la función *binary crossentropy* como función de coste para el entrenamiento. Se finaliza lanzando la ejecución del proceso de entrenamiento, con sendos generadores *data augmentation* para los conjuntos de entrenamiento y validación. El tamaño del batch es el predeterminado, de 32 elementos por lote, lo que nos deja un total de 1328 *steps / epoch*.

Asimismo, tal y como se incidió antes acerca de la tasa de aprendizaje (*lr*) adaptativa que utilizaba *custom-cnn*, en este caso también tenemos que hablar de la tasa de aprendizaje, y más concretamente, de una técnica empleada aquí y conocida como ***differential learning rate***, ligeramente distinta a la *adaptive learning rate* comentada antes.

Cuando hablamos de *adaptive lr* nos referimos a la variabilidad que puede sufrir la tasa de aprendizaje de una red a lo largo del tiempo, dependiente del número de épocas, y generalmente estamos dando por hecho algunos supuestos, como lo es el hecho de que toda la red, en todas sus capas y neuronas,

aplica el mismo valor de tasa de aprendizaje. En ese sentido, la variable es el tiempo o época en la que se encuentre la ejecución de la red, y no el elemento concreto sobre el que aplica dicha tasa.

Sin embargo, en el caso de esta red preentrenada, para la cual se ha reutilizado parte de la arquitectura original de *vgg16*, y se han congelado las capas reutilizadas, se aplica el concepto de *differential lr*, pues las capas congeladas tendrían un *lr* = 0, dado que no modifican los valores de sus parámetros, y por tanto no aprenden (ni desaprenden) nada nuevo. Sin embargo, el resto de la arquitectura, el nuevo grupo de capas añadidas en forma de capas extra sobre el modelo base, tendrá su *lr* específico, distinto al anterior, que como se comentó hace un momento, se trataba de un valor constante de 0.0004.

RESNET 50

Empezamos comentando el preprocesamiento realizado a los datos, y recordando lo comentado en el capítulo anterior, relativo al aumento de escala que se realiza a cada imagen. Aunque en este caso, dicho reescalado forma parte de la propia arquitectura de la red, no deja de ser una técnica de preprocesado de los datos, sea a través de la red, o a través de la modificación directa del dataset.

Nos referimos a las capas de reescalado (*UpSampling2D*) que van aumentando de forma progresiva el tamaño de cada imagen que entra a la red, pasando de los 32x32x3 píxeles iniciales, a los 256x256x3 requeridos por *resnet50* como formato de entrada.

El preprocesado se completa dividiendo el conjunto de datos inicial, como es habitual, en dos conjuntos, uno para entrenamiento y otro para validación y test. Ambos conjuntos son normalizados, dividiendo entre 255 imagen a imagen, cada una representada por una matriz de píxeles, y haciendo que los valores pasen de estar en el rango [0-255] al rango [0-1], mucho más conveniente computacionalmente.

Seguidamente, se codifican en formato *one-hot* los vectores de salida (Y) que contienen las etiquetas para las clases. Posteriormente, se crea un optimizador de tipo *RMSprop* con valor de *learning rate* igual a 0.00002, el cual se usa para compilar el modelo, junto con la función de coste *binary crossentropy*. El tamaño de batch para el entrenamiento es de 20 elementos por batch, lo que nos deja con un total de 2500 *steps / epoch*. El conjunto para validación, como se ha dicho, será el mismo que el de test.

5.3.2. Resultados

Se presentan aquí los resultados obtenidos, diferenciando a qué modelo corresponden.

VGGNET 16

Para el modelo de red que utiliza como base la arquitectura *vgg16*, con los parámetros especificados anteriormente, y refiriéndonos una vez más a su entrenamiento sobre el dataset objetivo del estudio, nos encontramos con los resultados que se muestran en la Figura 27.

Métrica / Época	#40	#100	#200	#500
Tiempo	6m 47s	16m 50s	33m 44s	1h 25m 44s
Accuracy	0.6861	0.7658	0.8282	0.8908
Loss	0.1494	0.1172	0.0895	0.0600
Val. Accuracy	0.6740	0.6907	0.6867	0.6796
Val. Loss	0.1522	0.1498	0.1706	0.2307

Figura 27. Resultados del entrenamiento de la arquitectura vgg16*.*
Fuente: elaboración propia

En este caso tenemos como mejor valor de *training accuracy* para todas las ejecuciones el obtenido después de 500 épocas, con un 89.08%, tras 1h 25m y 44s de entrenamiento.

Los valores para el *test accuracy* / *test loss*, ordenados por época, serían: **0.1206 / 258.3686** (#40), **0.1414 / 568.1662** (#100), **0.0903 / 1439.0708** (#200) y **0.0993 / 3588.0720** (#500).

Se ha decidido mostrar también los resultados de una prueba realizada por separado, en la cual no se utilizaban tres conjuntos de datos diferentes sino dos, uno para entrenamiento, con las 50 mil imágenes originales del dataset destinadas a esa fase, y otro empleado tanto para validación como para test, con los 10 mil ejemplos restantes. Se muestran los valores obtenidos tras 200 épocas en la Figura 28. En ella podemos observar que, usando el mismo conjunto de datos para validar y evaluar la red, los valores son acordes a los de la *test accuracy* y *test loss* del ejemplo con tres datasets.

Métrica / Época	#200
Tiempo	41m 6s
Accuracy	0.8188
Loss	0.0922
Val. / Test Accuracy	0.1109
Val. / Test Loss	1073.0383

Figura 28. Resultados del entrenamiento de la arquitectura vgg16*, con el dataset de validación igual al de test.*
Fuente: elaboración propia

Finalmente, se esbozan de forma breve los datos relativos al tamaño ocupado por el modelo y pesos de la red. Con un tamaño ligeramente superior al caso anterior, aunque igual de despreciable en términos de impacto en el requerimiento de recursos, tenemos que esta arquitectura puede llegar a ocupar unos 8 MB (~7.7 MB) de espacio en disco o memoria RAM. Pero, igual que en todas las ejecuciones

realizadas, el consumo de memoria principal de video, la RAM de la GPU, sí que ha sido elevado, con valores de entre 5 y 10 GB del total de 16 GB de los que dispone la tarjeta gráfica.

RESNET 50

Por último, en relación al modelo que usa como punto de partida la arquitectura *resnet50*, y una vez más, ejecutándose sobre *cifar10*, se han alcanzado los resultados mostrados en la Figura 28.

Métrica / Época	#5	#10	#20
Tiempo	1h 1m 24s	2h 2m 6s	3h 54m 2s
Accuracy	0.8572	0.9597	0.9815
Loss	0.2352	0.0808	0.0317
Val. / Test Accuracy	0.9291	0.9501	0.9442
Val. / Test Loss	0.1222	0.0419	0.0415

Figura 29. Resultados del entrenamiento de la arquitectura resnet50.
Fuente: elaboración propia

Aquí se debe prestar atención especialmente a la *accuracy* lograda durante el entrenamiento, con un mejor valor obtenido del 98.15%, el más alto de las tres arquitecturas, tras 20 épocas y un tiempo de 3h 54m y 2s. Además, en esta ocasión, dicho valor se ve respaldado por una también elevada *validation* / *test accuracy*, la cual consigue llegar al 94.42%

Como en el caso anterior, al emplear el mismo conjunto de datos para la validación y la evaluación del modelo, se hace innecesario repetir dichos valores por separado para el test, pues son los mismos que los logrados para la validación.

Como colación, se comentan los valores relativos al consumo de RAM o espacio en disco. Ahora sí, a diferencia de los casos anteriores, nos encontramos con un tamaño importante para el modelo y los pesos que debe manejar, pues en conjunto, los de esta arquitectura suman 156 MB.

Si bien sigue sin suponer un problema serio en lo referente a los recursos necesarios para ejecutar la red, es sin duda un tamaño notable, en especial si lo comparamos con los casos anteriores, donde apenas se llegaba a los 2 MB u 8 MB. Sin embargo, tiene una explicación muy sencilla, que se analizará en el siguiente capítulo, y es la enorme diferencia en la cantidad de pesos que manejan unas arquitecturas y otras. De nuevo, el consumo de memoria de video se situó en valores equiparables a los de las anteriores arquitecturas, con 10-12 GB consumidos, del total de 16 GB disponibles.

6. Discusión y análisis de resultados

Se dedicará este capítulo a la discusión acerca de los resultados obtenidos, presentando las principales conclusiones y extrayendo un significado de los mismos, hecho que nos permite sacar a la luz qué ventajas y desventajas presentan unos y otros.

Comenzamos recordando el *tamaño* de cada modelo, medido en número total de parámetros (*entrenables* / *no entrenables*) de cada una de las arquitecturas propuestas en la comparativa:

- **Arquitectura *custom-cnn***: 309.290 parámetros (*308.394* / *896*)
- **Arquitectura *vgg16***: 1.870.666 parámetros (*134.666* / *1.736.000*)
- **Arquitectura *resnet50***: 40.899.018 parámetros (*40.583.370* / *315.648*)

Tras estos números, cabe recordar un par de particularidades de las redes preentrenadas.

Mientras que para el caso del modelo *vgg16*, la arquitectura base original tenía 138 millones de parámetros, esta solo se ha reutilizado parcialmente, eliminando sus últimas capas, que habitualmente son densas, y cuentan con un mayor número de neuronas e interconexiones, y por tanto, de parámetros calculables. Después de ello, únicamente han quedado los casi 2 millones de parámetros totales del modelo final, pero, debido a que se han congelado las capas del modelo base, los parámetros entrenables ascienden a tan solo 134 mil, los correspondientes a las capas densas añadidas después del mencionado modelo base.

Por otro lado, para la arquitectura *resnet50*, no solo se ha reutilizado la arquitectura original por completo (excepto la última capa densa de salida con 1000 neuronas), la cual ya contaba con 23.587.712 parámetros, sino que se han añadido unas capas densas finales, que suponen los casi 17 millones de parámetros restantes, hasta alcanzar la suma anteriormente expuesta de 40 millones.

Con este breve apunte acerca de los parámetros, es posible concebir una idea del poder expresivo y de aprendizaje que puede llegar a alcanzar cada modelo, pues, en términos generales, a mayor número de parámetros entrenables, aumentarán los cálculos que debe realizar la red, y con ello, el tiempo de entrenamiento requerido para completar cada época y el proceso completo de entrenamiento. Sin embargo, a mayor cantidad de parámetros, también mayor capacidad tendrá la red para inferir y realizar predicciones ante ejemplos no vistos anteriormente.

A continuación, se procederá a hacer un análisis de los resultados, primero mostrando los gráficos con las puntuaciones logradas por cada red, tanto para la *accuracy* como para la *loss*, para finalmente agrupar todas las arquitecturas en dos gráficos globales con los valores alcanzados por cada una, de nuevo uno para *accuracy* y otro para loss, diferenciando cada modelo con una gama cromática distinta.

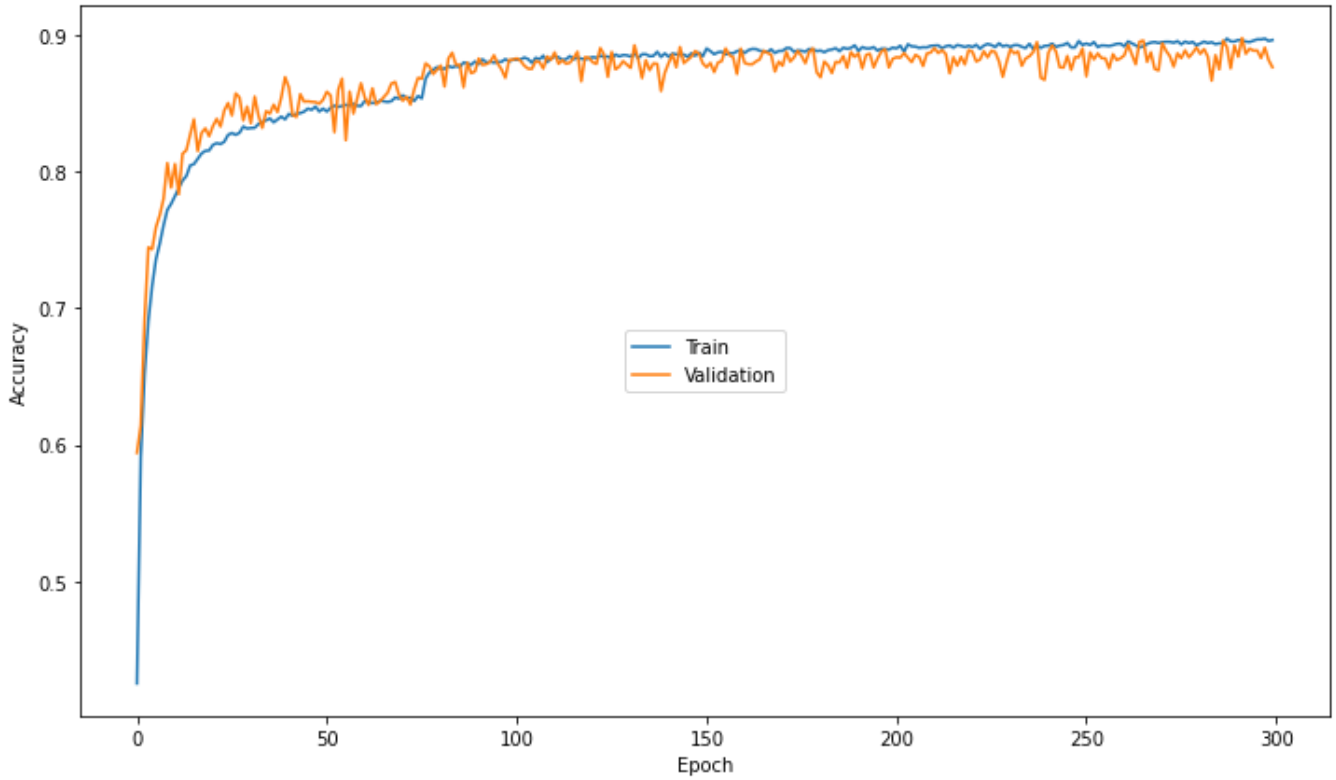

Figura 30. Gráfica con la accuracy *para* custom-cnn.
Fuente: elaboración propia

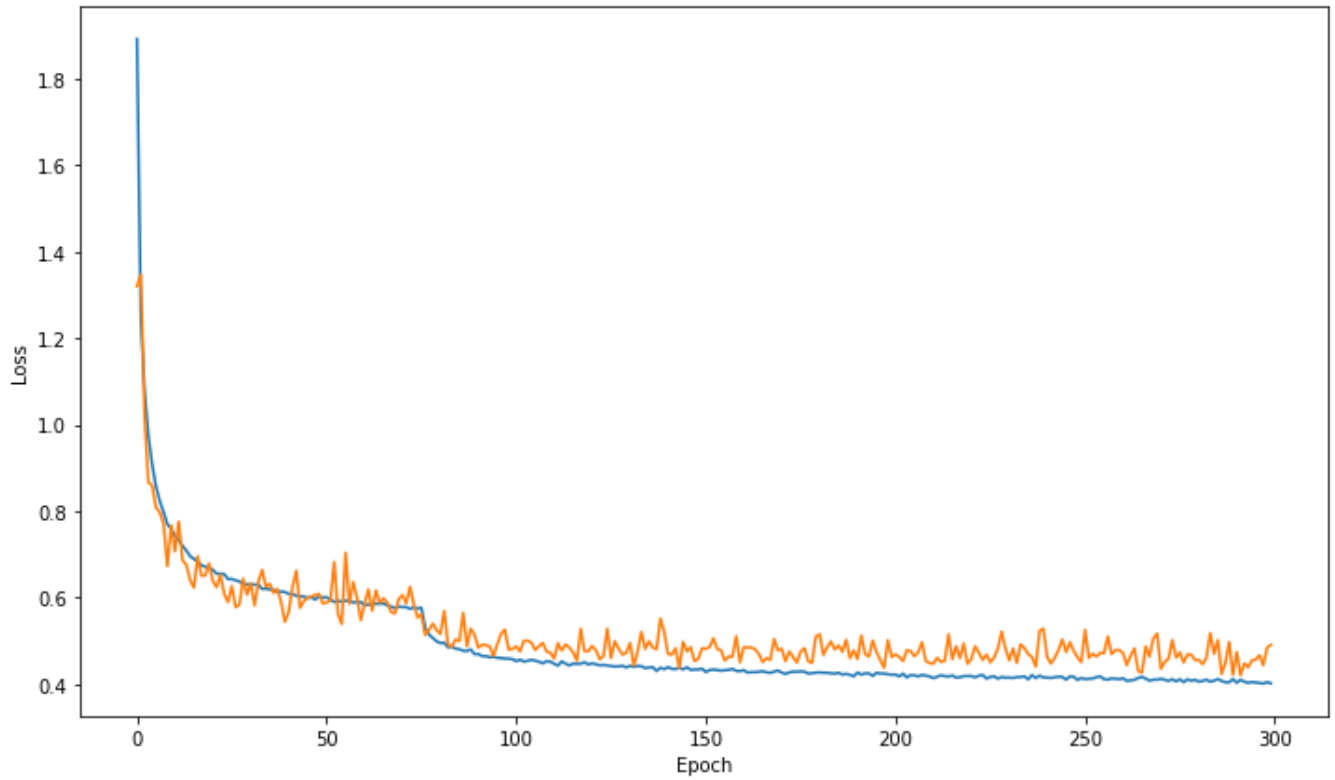

Figura 31. Gráfica con la loss *para* custom-cnn.
Fuente: elaboración propia

6.1. Custom CNN

Se comienza aquí mostrando los resultados de *accuracy (train & validation)*, y *loss (train & validation)*, para la red denominada *custom-cnn*, en la Figura 30 y la Figura 31, respectivamente.

Como podemos ver, la Figura 30 muestra el incremento en *accuracy* de forma progresiva, hasta alcanzar su cénit en la época 300, con un valor pico de 89.67%. Un par de detalles a comentar sobre esta imagen serían, por un lado, el acompañamiento gradual que realiza el valor para *validation accuracy*, aumentando también el valor de *training accuracy*, lo cual es un indicativo de que la red está aprendiendo de forma uniforme y constante, o al menos, de que el entrenamiento está siguiendo pautas normales.

Por otro lado, se puede apreciar un salto pronunciado en el valor de *accuracy* alrededor de la época 75, momento en el cual, precisamente, se produce una modificación de la tasa de aprendizaje de la red, como se comentó anteriormente, pasando de tener un valor de 0.001 a justamente la mitad, 0.0005. Debido a esto, la red comienza a afinar de forma más precisa sus resultados, aunque también lo hace de manera más lenta, pues la tasa de aprendizaje es menor.

Como era de esperar, de forma inversamente proporcional, vemos en la Figura 31 el descenso paulatino del valor de *loss* que va calculando el modelo, con una caída marcada al sobrepasar la época número 75, momento en el cual se procede a modificar la tasa de aprendizaje como se ha indicado antes.

Por tanto, en esta ocasión parece adecuada la idea de introducir una tasa de aprendizaje adaptativa, es decir, variable con el tiempo, pues produce resultados manifiestamente positivos, como los expuestos aquí. Sería incluso posible refinar aún más los valores, pero, al menos en la implementación utilizada para esta técnica, ello supondría tener que ajustar dichos parámetros manualmente.

6.2. VGGNet 16

Se mostrará en las siguientes cuatro figuras el rendimiento exhibido por el modelo *vgg16*. En primer lugar, se expone el caso para tres conjuntos de datos: entrenamiento, validación y test (Figura 32 y Figura 33). Después, se repetirá el proceso, pero en este caso, se presentará la ejecución para solamente dos conjuntos de datos diferentes: entrenamiento y validación/test (Figura 34 y Figura 35).

La motivación para proponer dos vías alternativas para la ejecución de este modelo se debe a que se entendió conveniente resaltar la utilidad del conjunto de validación, siempre opcional, en un caso como el que nos ocupa. Gracias a él, se pueden aplicar técnicas anteriormente citadas, como la de *early stopping*, permitiendo detener la ejecución de la red procediendo a su modificación en el momento en el que se detecta que esta no sigue mejorando para ejemplos no vistos previamente.

Es por ello que el uso del citado conjunto de validación se antoja especialmente útil para el propósito de ir afinando los hiperparámetros de una red hasta conseguir un modelo estable y robusto.

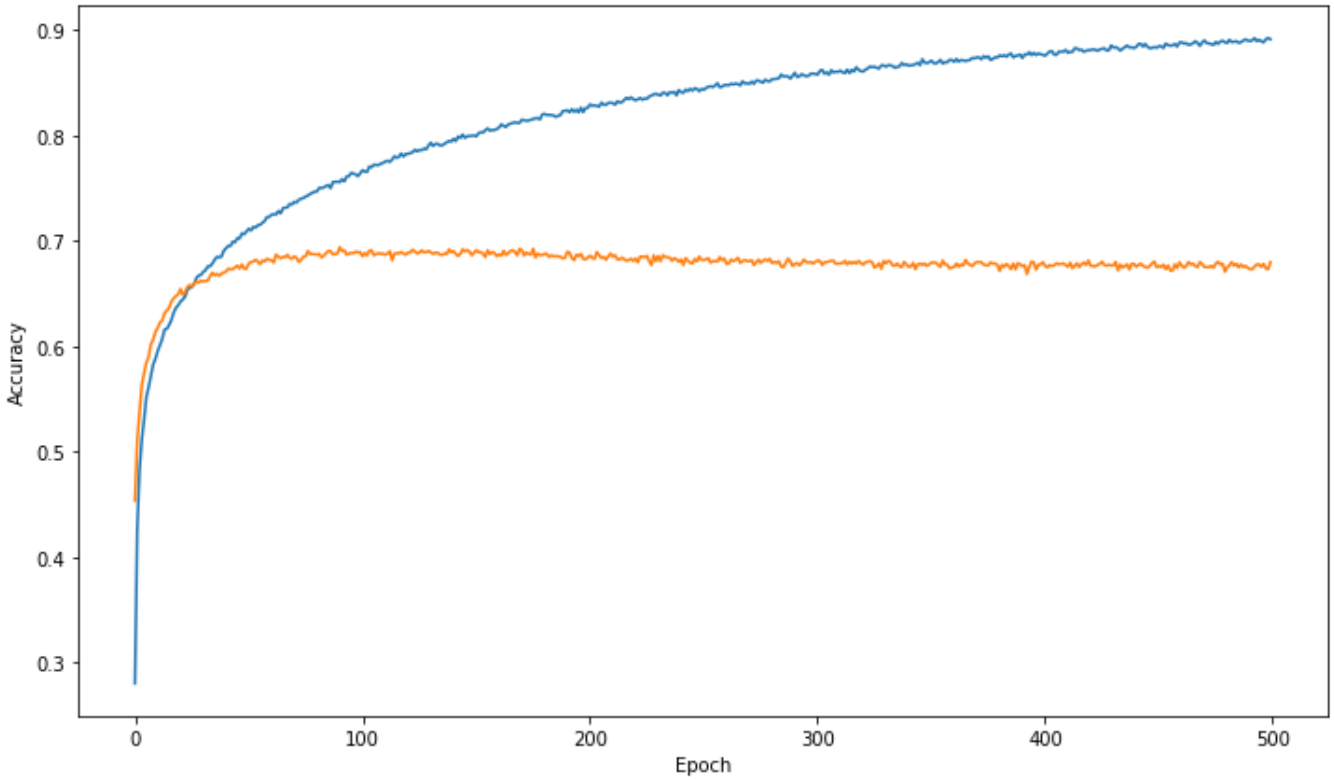

Figura 32. Gráfica con la accuracy *para* vgg16*, tres conjuntos de datos.*
Fuente: elaboración propia

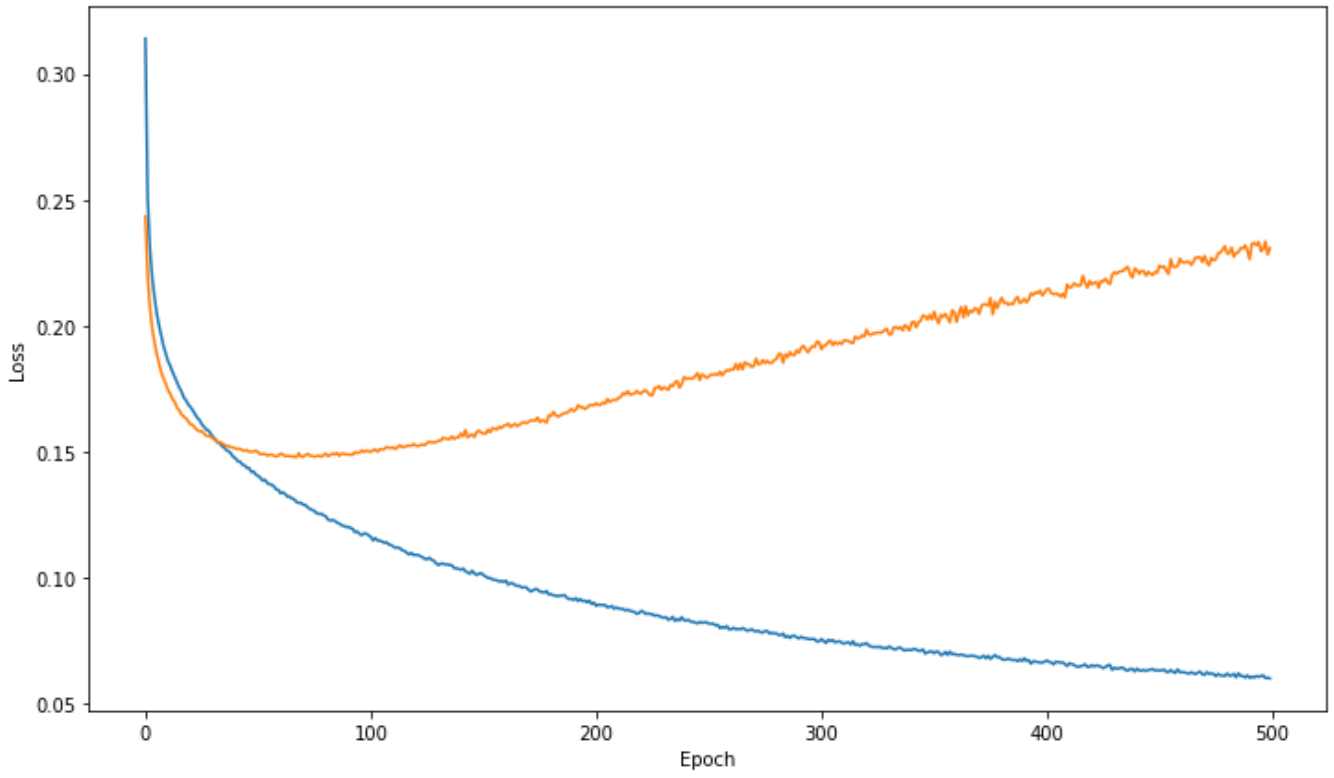

Figura 33. Gráfica con la loss *para* vgg16*, tres conjuntos de datos.*
Fuente: elaboración propia

Ciñéndonos a la primera situación, con tres datasets, vemos cómo en la Figura 32 se produce un estancamiento claro en los valores de *validation accuracy* de la red a partir de la época 70 u 80, aproximadamente. Sin embargo, la red sigue mejorando sus resultados para la *training accuracy* de manera sostenida incluso llegando a las 500 épocas ejecutadas. De hecho, si nos fijamos ahora en la Figura 33, vemos como la línea azul, en descenso para los valores de *training loss*, reafirma lo visto anteriormente con el incremento gradual de la *training accuracy*.

No obstante, la línea naranja, que marca los valores para la pérdida en el conjunto de validación, no solo para de reducir su valor (objetivo principal del optimizador, disminuir el valor de la función de coste), sino que se puede apreciar cómo comienza a crecer ligeramente de nuevo, indicando claramente que la red está empeorando sus resultados para la validación, mientras que progresa de forma positiva para el conjunto de entrenamiento.

Por otro lado, centrándonos en la ejecución con dos conjuntos de datos, que emplea uno de ellos tanto para la validación como para el test, obtenemos los resultados de la Figura 34 y de la Figura 35.

En primer lugar, se puede apreciar en la Figura 34 como el *training accuracy* se mantiene igual que en el caso anterior, pues la red aprende al mismo nivel que antes, si consideramos que en este caso, se han ejecutado menos épocas en total. Sin embargo, vemos cómo, ya sin ejecutar todas las épocas, la red está manteniendo un valor promedio muy pobre de *accuracy* para el conjunto de validación.

En segundo lugar, si nos fijamos en la Figura 35 vemos un valor disparado y en constante aumento para la *validation loss*, denotando el nulo aprendizaje que está consiguiendo realizar, y un valor plano y lineal para la *training loss*, que no se puede apreciar por la escala, pero se intuye similar al caso previo, con valores pequeños y reducción paulatina de los mismos.

Con los resultados de este modelo en la mano, tanto para el caso de dos como para el de tres conjuntos de datos, podemos confirmar lo siguiente, la red cae en ***overfitting***, no consiguiendo realizar un proceso de aprendizaje acertado, que le sirva para predecir la categoría de imágenes no vistas antes. Esto quiere decir que está siendo competente clasificando de forma correcta un porcentaje cada vez mayor de ejemplos del conjunto de entrenamiento, pero está mostrando resultados nefastos para los ejemplos no vistos del conjunto de validación, lo que revela su incapacidad para clasificar nuevas imágenes. En otras palabras, la red está aprendiendo de memoria, excesivamente bien, los datos que usa para entrenar, pero lo hace de forma tan específica, que pierde capacidad de generalización.

En este caso, el problema no reside tanto en los conjuntos de datos usados, o la distribución de los mismos, sino que se explica por la arquitectura del modelo, inadecuada para la tarea que se perseguía, lo que plantea la necesidad de realizar algunos cambios si se quiere construir algo realmente útil.

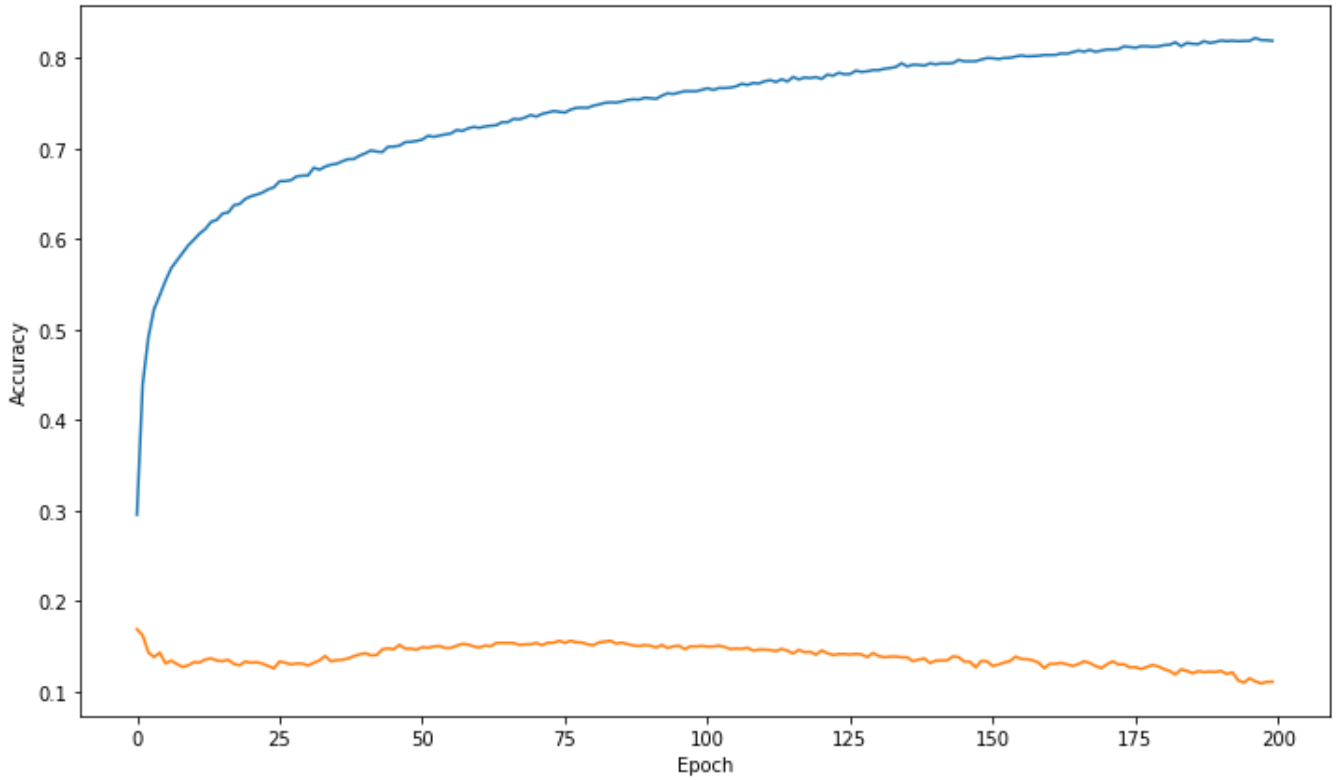

Figura 34. Gráfica con la accuracy *para* vgg16*, dos conjuntos de datos.*
Fuente: elaboración propia

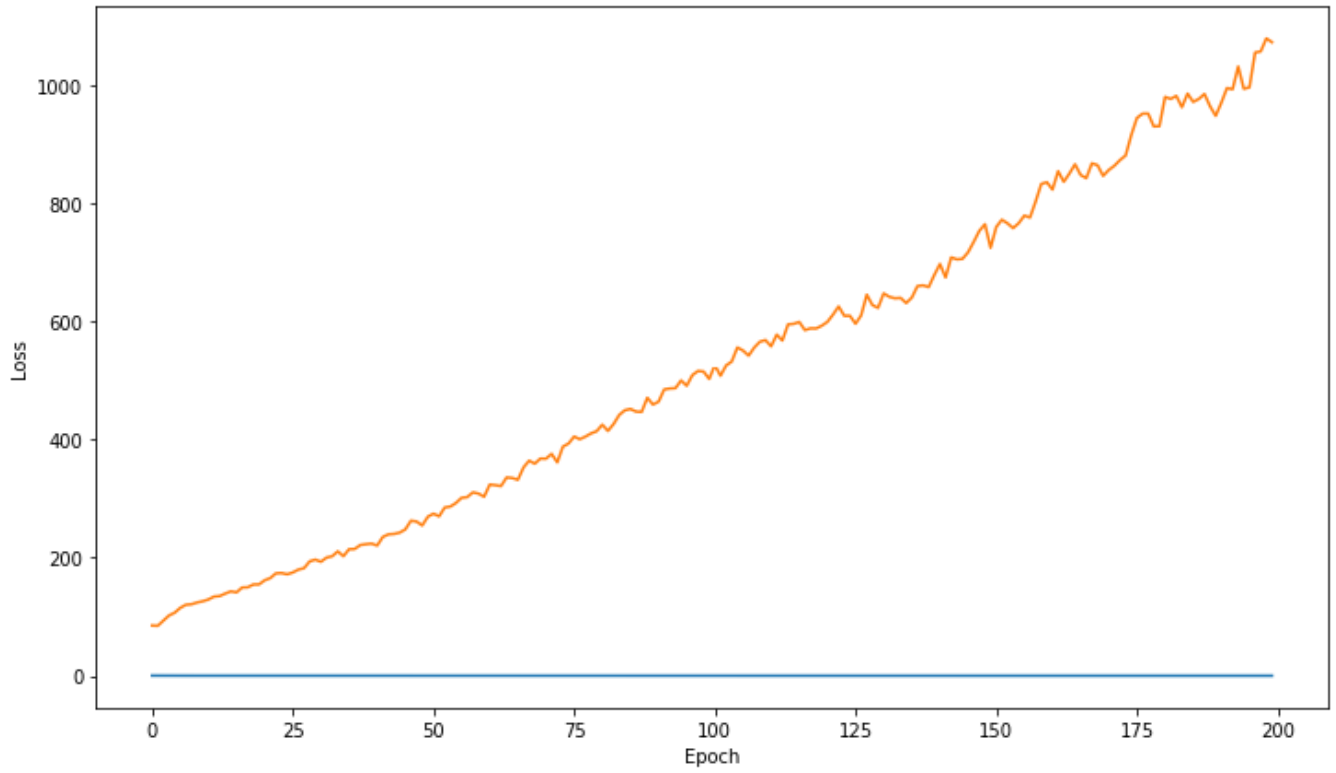

Figura 35. Gráfica con la loss *para* vgg16*, dos conjuntos de datos.*
Fuente: elaboración propia

D

En efecto, se realizaron varios intentos de ejecución modificando otros parámetros, que no se muestran detalladamente aquí para no alargar demasiado la explicación. Se probó modificando la función de pérdida empleada (*categorical crossentropy*, en lugar de *binary*), y el optimizador (probando con *RMSprop* y diferentes valores de tasa de aprendizaje, en vez de utilizar *Adam*), pero en todos los casos el desempeño mostrado era igual de insatisfactorio.

Se aportan aquí un par de razones que pueden haber conducido a un resultado como este:

- Primeramente, se ha reutilizado solo parte de la arquitectura del modelo base original de VGGNet 16, descartando los dos últimos bloques convolucionales, y manteniendo únicamente los tres primeros. Tengamos en cuenta la forma de aprender de las redes profundas, partiendo de conceptos simples (formas sencillas, como líneas o bordes, cuando hablamos de redes convolucionales), y sobre ellos construyendo otros más complejos, a medida que avanza la profundidad de las capas. Debido a ello, es posible que se hayan eliminado demasiadas capas convolucionales del modelo base, dejando una profundidad insuficiente para que la red sea capaz de disgregar de forma apropiada las características de cada imagen. Por tanto, las últimas capas densas ven afectada su efectividad, al trabajar sobre características irrelevantes, o representadas de formas erróneas.
- Por otro lado, todas las capas del modelo base utilizadas han sido congeladas, impidiendo que puedan entrenarse de nuevo y aprender conceptos diferentes a los ya adquiridos. Además, esto puede haber dificultado el hecho de que la red extraiga características nuevas de las imágenes, pues no solo tiene una escasa profundidad de capas, sino que además, las únicas capas convolucionales de las que dispone, las encargadas de esa extracción de características, se ven inhabilitadas para hacerlo. En este sentido, el modelo no podrá hacer mucho ante nuevos ejemplos o variaciones sustanciales en los mismos. En resumen, no representará bien lo que está percibiendo (las características de las imágenes), y por tanto, las capas densas no podrán trabajar sobre ellas de manera apropiada, haciendo que la red realice predicciones en su mayor parte erróneas.

En conjunto, tanto el hecho de no disponer de suficientes capas convolucionales, como el hecho de que tenga una profundidad tan reducida que le impida desgranar adecuadamente la complejidad de los conceptos, han hecho que la utilidad real de la red, en la práctica, sea nula. Por muy bien que entrene y mejore sus resultados sobre el conjunto de entrenamiento, la mayor parte del tiempo seguirá siendo incapaz de predecir de forma correcta los nuevos ejemplos, siendo ese uno de los objetivos perseguidos.

6.3. ResNet 50

Por último, empezamos exponiendo los resultados cosechados por la arquitectura perteneciente al modelo *resnet50*, que es el que ha exhibido con diferencia un mejor resultado de entre todas las soluciones planteadas y formuladas en el presente trabajo.

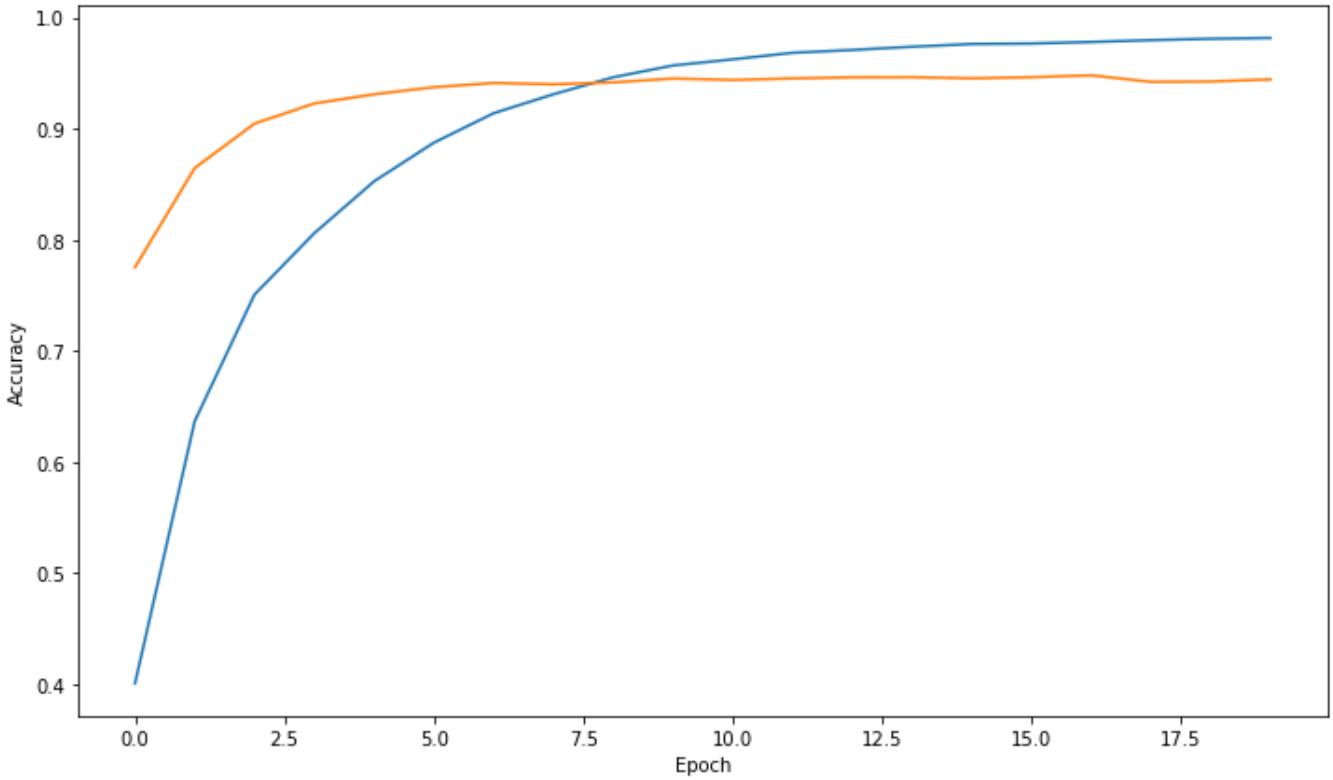

Figura 36. Gráfica con la accuracy *para* resnet50.
Fuente: elaboración propia

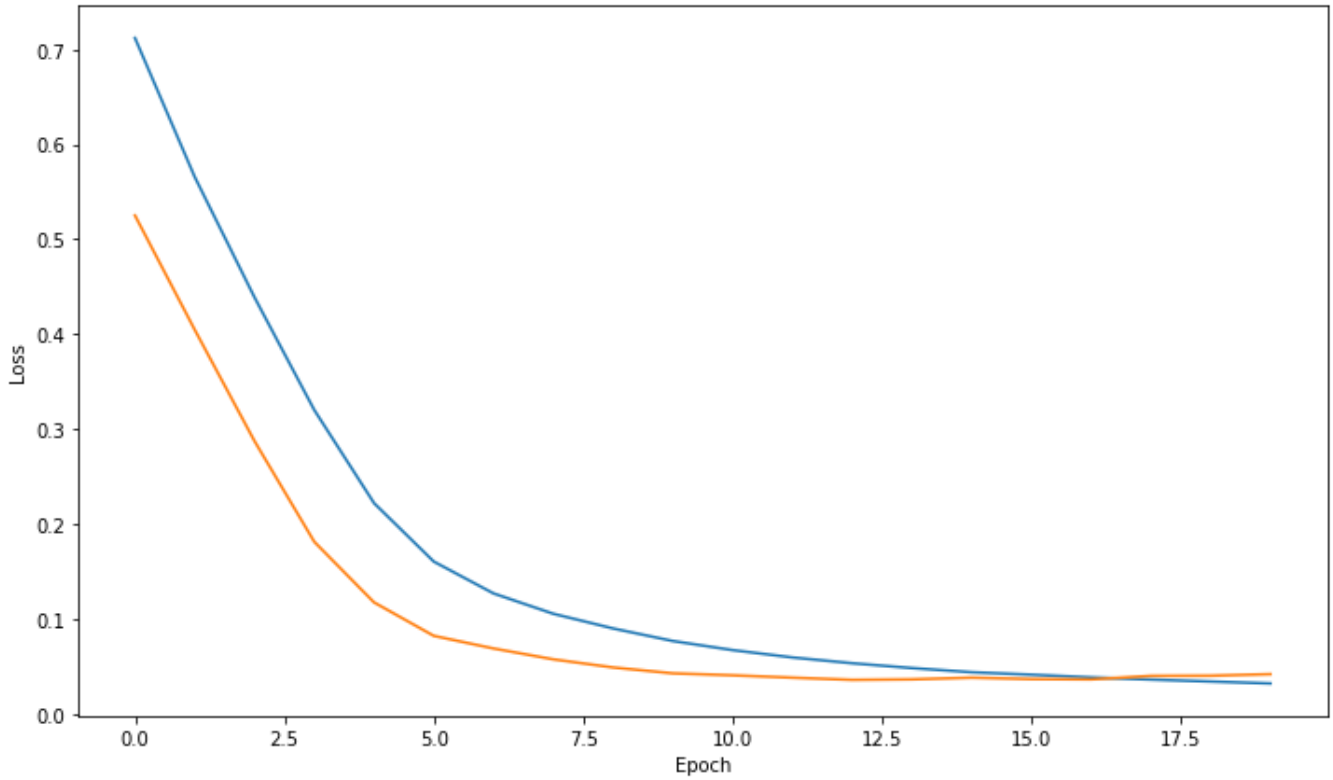

Figura 37. Gráfica con la loss *para* resnet50.
Fuente: elaboración propia

Se muestran los datos relativos a su *accuracy* y *loss* en la Figura 36 y la Figura 37, respectivamente. Se trata de la red que menor número de épocas ha precisado para converger de manera satisfactoria, alcanzando un *accuracy* del 98.15% tras únicamente 20 de ellas ejecutadas. Eso sí, a pesar de haber empleado solamente esa pequeña cantidad para ese valor de *accuracy*, ha necesitado casi 4 horas de entrenamiento, pues el tiempo promedio por época ha sido elevado (11m 40s), debido al gran tamaño de la red, y los más de 40 millones de parámetros a calcular en cada iteración.

Observando las imágenes nos encontramos, en la Figura 36, con una curva de progreso bastante buena para la *training accuracy*, alcanzando valores ya altos desde la segunda o tercera época. Por otro lado, el valor para la *validation accuracy*, igual que en el caso del modelo *custom-cnn*, acompaña de forma razonable. Si pasamos a focalizar nuestra atención en la Figura 37, se ratifica la misma idea, pues se produce un descenso equivalente en los valores de *training loss*, seguidos por los de la *validation loss*.

Ambas gráficas sugieren un par de ideas muy interesantes acerca de su comportamiento, según nos encontremos al inicio, en las primeras épocas, o al final, en las últimas:

- Por un lado, recordemos que esta arquitectura aplica un par de capas de dropout agresivas (50% de inactivación) a continuación del par de capas densas añadidas después de su modelo base original. Eso podría provocar que la red sufriera a la hora de aprender conceptos, pues una vez adquiridos, es posible que no pudiera utilizarlos en la siguiente etapa, debido a la desactivación aleatoria de ciertas neuronas, y se viese forzada a buscar otras vías para representar esa característica, pudiendo redundar en soluciones incluso mejores. La consecuencia directa es que, mientras a lo largo del entrenamiento se utiliza la red de manera parcial (desactivando neuronas, y con ello, características aprendidas), en las fases de validación o evaluación se utiliza el poder expresivo de la red al completo, pues no se aplica ningún tipo de dropout.

  Debido a esto, se puede apreciar cómo al inicio, hasta que la red alcanza la robustez necesaria, aprendiendo las suficientes vías para representar las características de las imágenes, obtenemos mejores resultados en la validación que en el entrenamiento. Es decir, inicialmente sabe categorizar mejor incluso ejemplos no vistos, que aquellos que ya ha procesado.

- Por otro lado, también se puede apreciar un ligero *overfitting* hacia la parte final de ambas gráficas. En ellas se distingue cómo, desde aproximadamente la mitad del camino, se produce un ligero estancamiento en los conjuntos de validación, tanto para el incremento que se venía observando en la *accuracy*, como para la reducción progresiva marcada por la *loss*.

  Sin embargo, en el caso de los conjuntos de entrenamiento, la *accuracy* mantiene su ascenso constante, cercano al 100%, y la *loss* tiende a 0. Por este motivo, se alcanza un punto de inflexión, en el que la *training accuracy* supera y mantiene un crecimiento sostenido respecto a la *validation accuracy*, y la *training loss* reduce sus valores por debajo de la *validation loss*.

Es decir, la red sigue mejorando, pero solo para su conjunto de entrenamiento, lo cual podría ser un buen indicio para detener la ejecución de manera temprana, y realizar las modificaciones oportunas, en caso de ser necesario. O, por el contrario, considerar el modelo como suficientemente robusto en algún punto anterior a que se produzca el mencionado *overfitting*, y tomar como aceptable un valor de *accuracy* inferior al máximo que podría obtener la red. A cambio de ello, se garantiza una mejor capacidad de generalización ante nuevos ejemplos.

6.4. Comparativa global

Presentamos aquí el apartado final de este capítulo dedicado al análisis de los resultados, donde se desglosarán de forma comparativa algunas ideas globales acerca de las distintas soluciones desarrolladas. Para ello nos valemos de dos gráficas (*accuracy* y *loss)* que, por cuestiones de escala, se han dimensionado en dos rangos de épocas diferentes, uno más ampliado que el otro, haciendo un total de cuatro imágenes. Se muestran en la Figura 38 (*accuracy* 0-200) y la Figura 39 (*accuracy 0*-20), así como las correspondientes Figura 40 y Figura 41, para *loss*.

Como se mencionó antes, a cada red se le asigna una gama cromática, especificada aquí a modo de aclaración, siendo la primera tonalidad la del conjunto de entrenamiento, y la segunda la de validación:

- *Custom CNN* (**azul / azul claro**)
- *VGGNet 16,* tres conjuntos de datos (**rojo / marrón**)
- *VGGNet 16,* dos conjuntos de datos (**negro / gris**)
- *ResNet 50* (**verde oscuro / verde lima**)

Comenzamos hablando de las Figuras 38 y 39. En la primera, la cual proporciona la visión más amplia del conjunto de resultados, podemos ver en último lugar los modelos *vgg16*, tanto con dos como con tres conjuntos de datos. Vemos que sus curvas de *training accuracy* son bastante similares, aunque se diferencian de forma notable en sus curvas de validación. El modelo que usa como conjunto de validación una parte del conjunto de entrenamiento, obtiene mejores resultados, pero alcanza pronto un estancamiento alrededor del valor 70% de *validation accuracy*. Para el caso de la ejecución que utiliza el conjunto de test para la validación, vemos unos resultados claramente inferiores.

Les sigue, con un buen rendimiento exhibido, el modelo *custom-cnn*, que sin llegar a alcanzar lo logrado por la mejor de las arquitecturas, consigue el segundo mejor rendimiento del estudio, con unas curvas de aprendizaje razonables, y coherentes entre sus fases de entrenamiento y validación. Ya dentro de las redes que parten de cero, su 89.67% de *accuracy* le posiciona como una buena solución.

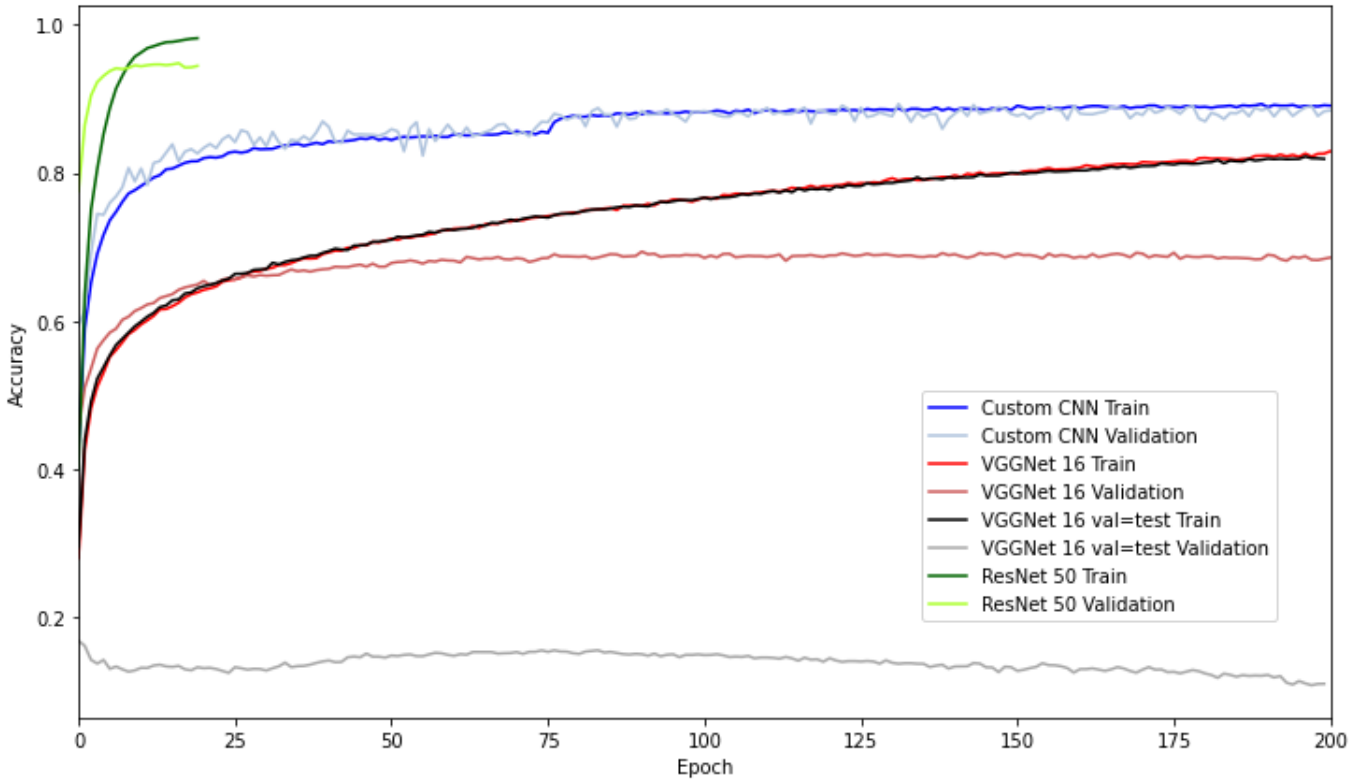

Figura 38. Gráfica con la accuracy *global (época 0-200)*
Fuente: elaboración propia.

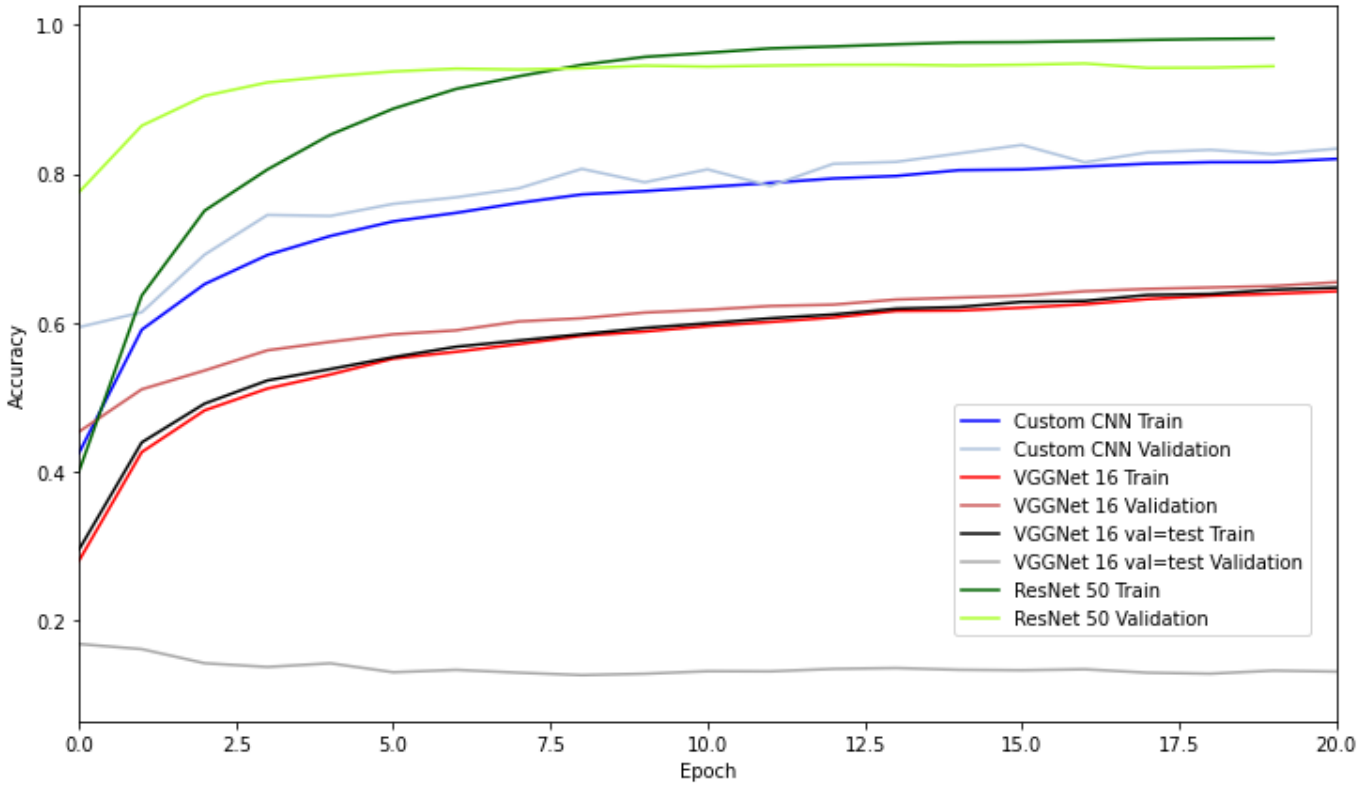

Figura 39. Gráfica con la accuracy global (época 0-20).
Fuente: elaboración propia

Por último, si nos fijamos en la arquitectura *resnet50*, para lo cual conviene irnos a la Figura 39, se aprecia cómo esta destaca claramente, obteniendo los mejores resultados de todas las arquitecturas propuestas. En esta última figura se puede ver de forma inequívoca la posición que ocupa cada una según su rendimiento, con una posición alta para *resnet50*, en tonalidades verdes, una posición media para *custom-cnn*, con matices azules, seguidas ambas, en las posiciones más bajas, por las ejecuciones de los modelos *vgg16*, con los colores rojo, para tres conjuntos, y negro, para dos conjuntos de datos.

Se pueden percibir ideas similares cuando nos fijamos en las Figuras 40 y 41, para la métrica *loss*. Como se puede observar, en dichas figuras no aparece el valor de la *validation loss* para el modelo *vgg16* con dos conjuntos de datos, pues sus números se disparaban de tal forma que descuadraban el resto de la gráfica, además de contar con poca relevancia, por lo que se ha decidido evitar dicha medida.

Una vez más, *resnet50* es la que consigue una mayor reducción en su función de coste, mientras que, curiosamente, en este caso es la arquitectura *custom-cnn* la que menos logra decrementar su pérdida, tanto durante el entrenamiento como la validación, y cuenta por tanto con los mayores valores. De esta forma, se pone de manifiesto el hecho de que se pueden manejar valores de coste, dentro de unos límites, más altos que otra arquitectura, y aun así obtener mejor *accuracy* que ella.

Es la situación que nos encontramos con los modelos *vgg16*, pues, en sendas ejecuciones, los valores para la pérdida se han conseguido reducir de forma más significativa que para *custom-cnn*, y sin embargo, como ya hemos visto y comprobado hace unos párrafos, eso no ha significado un mejor rendimiento, al menos cuando hablamos de poner el modelo a funcionar ante ejemplos no vistos.

Con toda esta información, podemos concluir que ambas aproximaciones, tanto la de implementar y entrenar modelos completamente desde cero, como la de reutilizar redes preentrenadas, de forma total o parcial, son válidas de cara a conseguir unos objetivos aceptables sobre CIFAR 10, el dataset objetivo.

Resalta, de manera patente, que no siempre el simple hecho de utilizar redes preentrenadas y aplicar la técnica de *transfer learning* es suficiente para alcanzar buenas metas, sino que ese proceso se debe realizar con sumo cuidado, adaptando las particularidades de cada problema. Se debe considerar detenidamente qué partes de la arquitectura original se quieren conservar, cuales no, así como también tomar la decisión acerca de qué porcentaje de las partes reutilizadas será o no será entrenable.

Por otro lado, se debe tener en cuenta que el *transfer learning* es capaz de ofrecer mejores resultados cuando el nuevo problema al que se enfrenta la red preentrenada, aunque distinto, tenga similitudes con el problema sobre el que se entrenó originalmente. De lo contrario, el preentrenamiento podría carecer de utilidad, al no disponer la red de un aprendizaje suficiente en el nuevo ámbito en el que se ve envuelta, obteniendo casi el mismo resultado que si se entrenase completamente desde cero. Así pues, se debe intentar lograr un equilibrio entre los resultados ofrecidos por un modelo, y la facilidad de construirlo.

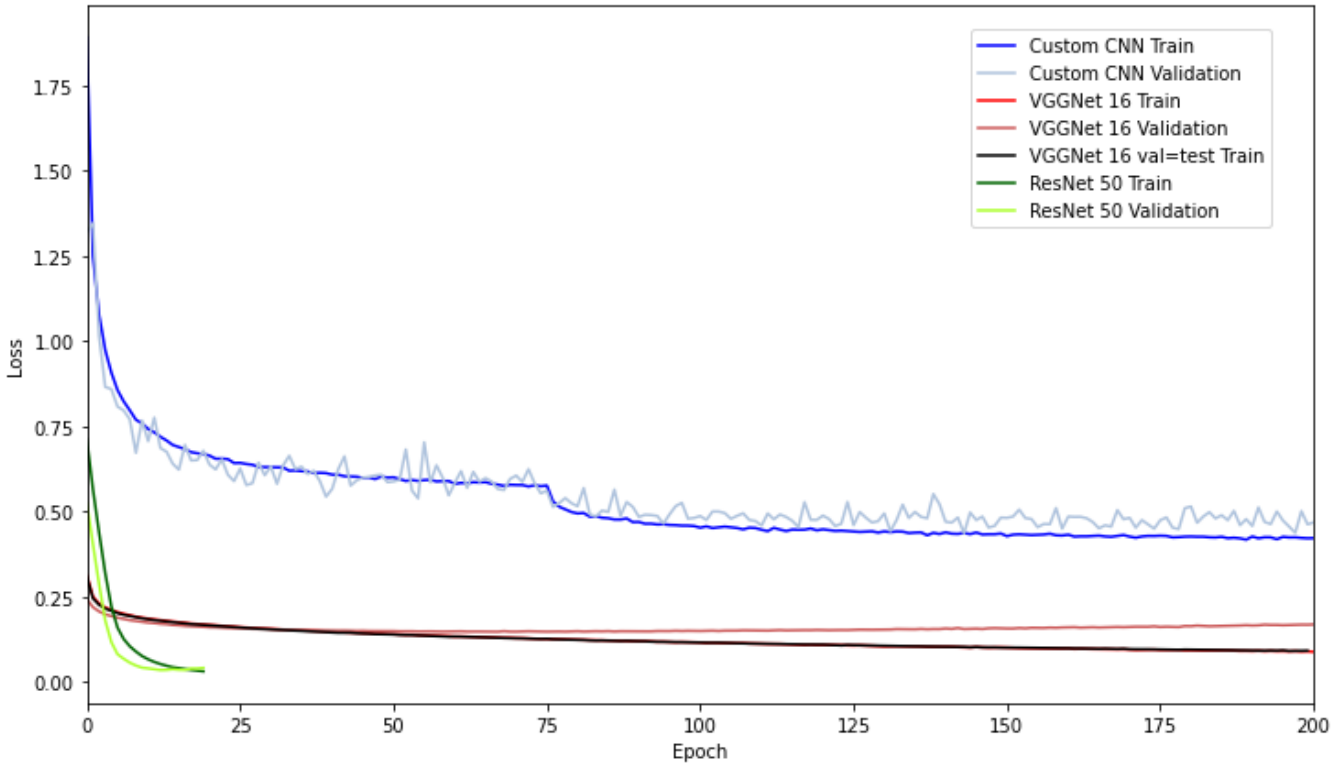

Figura 40. Gráfica con la loss global (época 0-200).
Fuente: elaboración propia

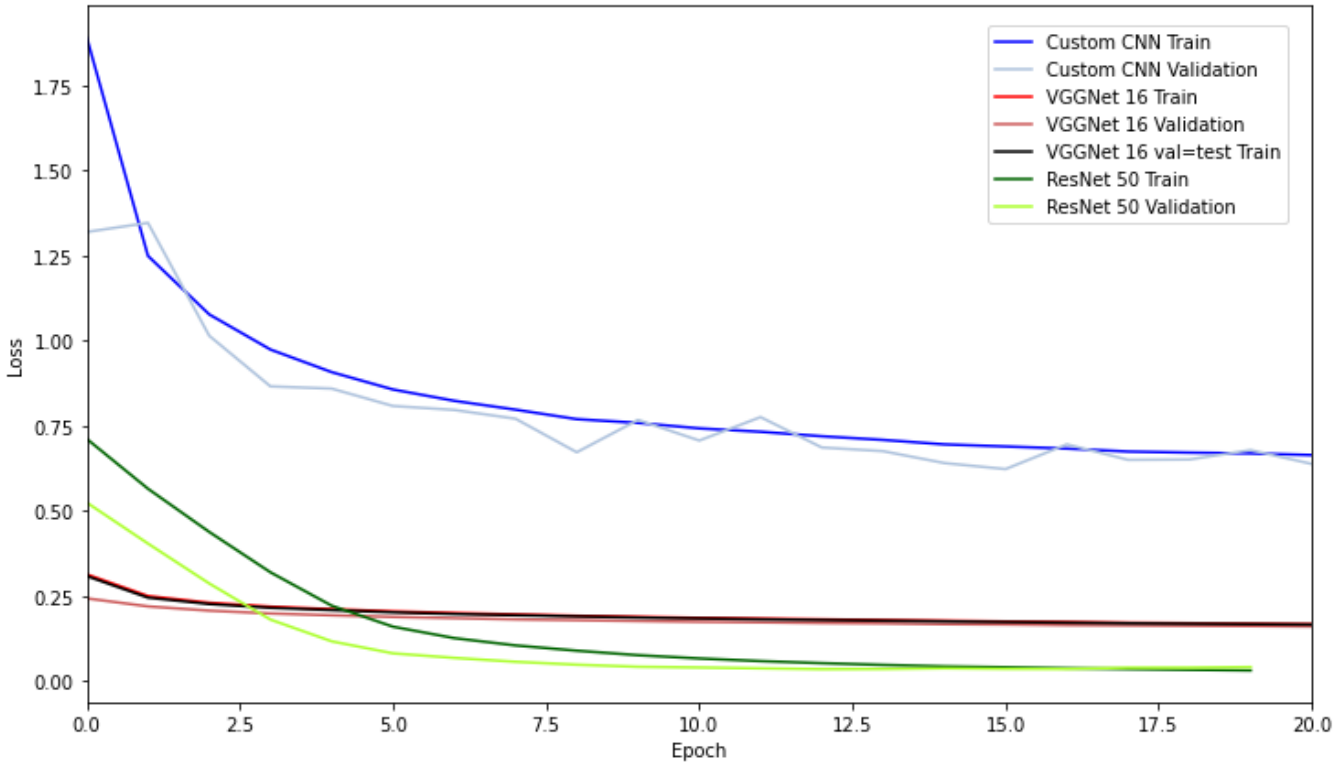

Figura 41. Gráfica con la loss global (época 0-20).
Fuente: elaboración propia

7. Conclusiones

A lo largo del presente trabajo se ha realizado un análisis comparativo de diferentes arquitecturas de redes neuronales convolucionales, las cuales copan desde hace unos años el estado del arte, en cuanto al nivel de rendimiento que puede llegar a ofrecer cualquier técnica aplicada en tareas relacionadas con la visión artificial, como el ejemplo que nos ocupa aquí, la clasificación de imágenes.

El propósito ha sido el de encontrar cuáles de estos modelos permiten alcanzar mejores resultados, qué diferencias existen entre ellos, y por tanto, qué ventajas y desventajas aporta cada uno. Ha sido posible obtener conclusiones relevantes acerca del estudio, que permiten ayudar a entender mejor qué arquitecturas pueden ser mejores respecto a otras, y qué técnicas y métodos pueden optimizar las metas alcanzadas por cada una, acelerando su convergencia.

Con ese guion, se ha abordado la mencionada comparativa, comenzando por un minucioso análisis del estado del arte en cuanto a redes convolucionales se refiere, haciendo breves repasos históricos por algunos conceptos de la IA, dentro de campos como el aprendizaje automático, el aprendizaje profundo, o el propio campo de la visión artificial. Tras ello, se han repasado las principales arquitecturas de redes convolucionales de los últimos años, muchas de ellas ganadoras de competiciones relacionadas con tareas propias de la visión artificial, como el ILSVRC.

Se ha proseguido el trabajo recapitulando algunas conclusiones acerca del análisis del estado del arte realizado, como el hecho de que existen multitud de modelos y arquitecturas, así como una amplia variedad de técnicas y métodos, dentro de las redes convolucionales, no siendo ninguna de las alternativas la mejor para todos los casos. Así pues, la decisión de elegir una u otra opción es en ocasiones compleja, y requiere de algunas pistas que ayuden a entender bien la problemática a enfrentar. En muchas ocasiones, el rendimiento o la calidad alcanzada por la solución desarrollada dependerá de adaptar varias de estas arquitecturas o técnicas al problema concreto.

Lo anterior desembocó en la idea central de efectuar un análisis que comparase y evaluase el rendimiento que ofrecían algunas de las diferentes arquitecturas descubiertas antes, centrando el alcance en torno a las redes neuronales especializadas en la clasificación de imágenes. Además de ello, también se plantearon unos objetivos específicos que perseguían razonar detenidamente acerca de los resultados presentados por el entrenamiento de varios modelos diferentes de estas redes convolucionales, ideando una comparativa de dos grupos o paradigmas de arquitecturas, aquellas que se implementan capa a capa y se entrenan desde cero, y aquellas que se basan en la reutilización de modelos ya entrenados, con el fin de aprovechar dicho aprendizaje adquirido.

Recordando estos objetivos:

1 *Seleccionar los modelos, datasets y librerías más adecuadas a utilizar en el estudio, justificando los motivos de su elección frente a otras alternativas, a partir de la revisión del estado del arte realizada en el capítulo anterior.*

Se comenzó haciendo un proceso de selección entre el conjunto de modelos, datasets y librerías descubierto gracias al análisis previo del estado del arte, estudiando qué alternativas podían encajar de mejor forma aquí, y justificando su elección en cada caso.

2 *Encontrar las mejores implementaciones disponibles para las arquitecturas de redes CNNs seleccionadas, y los formatos más eficientes para los datasets, así como las técnicas de preprocesado de los datos o entrenamiento del modelo óptimas para acelerar el proceso.*

Más tarde, una vez escogidas las arquitecturas y herramientas con las que se iba a afrontar el desarrollo, se realizó un estudio más técnico, para elegir qué implementaciones podían ser las óptimas, dentro de las disponibles para cada arquitectura convolucional seleccionada, y qué técnicas o métodos era conveniente aplicar en cada situación, bien para preprocesar los datos, entrenar los modelos, o para mostrar los resultados obtenidos por las sucesivas ejecuciones, mediante tablas o gráficas.

3 *Desarrollar el análisis comparativo, ejecutando y evaluando los modelos elegidos anteriormente, recolectando para ello diferentes métricas a utilizar en el análisis posterior.*

Una vez adquirido ese conocimiento acerca del dominio en el que se enmarcaba la comparativa, se desarrolló la misma, ejecutando de manera ordenada y consecutiva las diferentes configuraciones para cada modelo y conjunto de hiperparámetros, así como recolectando una serie de métricas de diferente naturaleza, que serían utilizadas más tarde para discutir los resultados.

4 *Discutir los resultados, tratando de interpretarlos y darles un significado, además de estudiar las ventajas y desventajas de las distintas opciones, extrayendo conclusiones relevantes.*

Por último, se concluyó esa parte del trabajo con la explicación e interpretación de los datos obtenidos como producto del desarrollo previo de la comparativa, analizando punto por punto cada caso mediante tablas y gráficas que mostrasen el comportamiento para cada modelo y métrica utilizada. A raíz de ello, se pudo dar un significado a tales datos, exponiendo las ventajas y desventajas que presentaban unas arquitecturas respecto a otras, extrayendo así conclusiones relevantes del estudio, las cuales pueden ser de utilidad como experiencia acumulada para futuros problemas de similar índole.

Tras estas líneas, resulta factible afirmar dos cosas. Por un lado, se cree satisfecho el objetivo general del presente trabajo, pues se ha planteado, desarrollado y debatido acerca del análisis comparativo ideado desde un inicio entre diversas arquitecturas de redes centradas en la clasificación de imágenes.

Por otro lado, también se han cumplido los objetivos específicos, como se ha ido repasando uno a uno en los últimos párrafos, gracias especialmente a las ideas descubiertas o razonadas como resultado del

análisis de las métricas recogidas. Estas ideas suponen pequeños aportes a futuros desarrollos o proyectos relacionados con el tema sobre el que ha versado el estudio, la construcción de redes neuronales profundas con buen rendimiento en tareas de clasificación de imágenes.

Finalmente, queda comentar, que si bien se consideran superados los objetivos planeados desde un comienzo, y alcanzada la idea sobre la que se deseaba desarrollar la investigación, quedan algunas sombras en el horizonte, limitaciones encontradas que imposibilitaron un mejor desenlace en los datos. Especialmente las relativas a la arquitectura *vgg16*, que aunque se esperaba que no ofreciese el mejor rendimiento de todas, si se consideraba al menos una opción viable que pudiera ofrecer resultados aceptables. Pero como se ha comprobado y aclarado, no ha sido el caso. Es por ello que queda como una de las tareas pendientes, que, entre otras, se comentarán en la siguiente sección.

8. Líneas de trabajo futuro

Se compartirán aquí algunas reflexiones acerca de qué mejoras se podrían haber realizado en el presente estudio, ya desde un comienzo, o *a posteriori*, viendo los resultados alcanzados.

Por un lado, la comparativa y el análisis previo realizados, dejan clara la enorme variedad de alternativas de las que disponemos dentro del terreno de las redes neuronales convolucionales, que representan sin duda el presente y futuro, al menos cercano, en el ámbito de las tareas relacionadas con la visión artificial. Esto abre la puerta a escalar el estudio en diferentes direcciones.

Desde cierto punto de vista, tendría sentido enfrentar entre sí un mayor número de arquitecturas, algunas de ellas ni siquiera mencionadas en el presente trabajo, para lo cual sería interesante explorar distintas alternativas al paradigma convolucional expuesto hasta ahora.

Por ejemplo, tendríamos las brevemente citadas redes generativas antagónicas (GAN), las cuales tienen mucho que decir en el campo de creación de ideas innovadoras desde un punto de vista computacional, a la hora de generar novelas u obras artísticas. Tales redes podrían suponer, y suponen ya en algunas investigaciones y equipos de trabajo, un complemento idóneo para las redes convolucionales, pues permite que las primeras (las GAN) generen ejemplos nuevos de manera continua para entrenar a las segundas (las CNN). Asimismo, la mejora paulatina que experimentan las CNN, hará que se conviertan en un filtro cada vez mejor de las creaciones generadas por las GAN, retroalimentándolas de nuevo, y haciendo que progresen a su vez en un ciclo constante semi-supervisado.

En contraste, cabe mencionar aquí otra aproximación al campo de la visión artificial realizado por arquitecturas denominadas como redes de cápsulas, o *CapsNets* (Sandu & Karim, 2020), las cuales pretenden suplir o completar los modelos de redes CNNs. Intentan hacerlo superando algunas de las limitaciones conocidas de las arquitecturas convolucionales, como el hecho de que estas son buenas para detectar características en las imágenes, pero no tanto para ubicar de manera relativa tales características, pudiendo sufrir ante rotaciones de las mismas.

Este último par de ejemplos muestran a la perfección hacia donde podría dirigirse una futura ampliación de esta comparativa, pues diversidad de arquitecturas, paradigmas y técnicas, así como de datasets, no faltan. A pesar de que, con la intención de salvaguardar su alcance, en este trabajo nos hemos ceñido a un solo dataset, justificado como suficiente, se podría haber añadido alguno de los otros mencionados, para verificar las diferencias de comportamiento según el conjunto de datos utilizado para aprender.

Entre ellos, podríamos haber utilizado CIFAR 100, si la intención fuera exprimir al máximo y poner a prueba, a gran escala, cada modelo, dado que se trata de un conjunto muy similar al utilizado aquí, CIFAR 10, pero amplificado con numerosas categorías y ejemplos más.

Haciendo un repaso por los resultados obtenidos, otra de las posibles mejoras que se podrían realizar sería la de estudiar detenidamente la arquitectura del modelo *vgg16*, examinando las causas concretas que le llevaron a tan mal rendimiento, y planteando las modificaciones oportunas que podrían conducir a que manifestase un desempeño al menos igual de bueno que el de los otros modelos analizados aquí.

Para ello, se podría reconstruir al menos parte del modelo, añadiendo profundidad a la red, con un mayor número de capas convolucionales, haciendo que esta sea capaz de inferir y representar conceptos con cierta utilidad, de cara a un correcto funcionamiento. Asimismo, una vez arreglada la arquitectura, sería conveniente explorar diferentes combinaciones de alternativas relativas a qué técnicas aplicar, qué optimizador es más beneficioso utilizar, o que inicialización seguir con los parámetros, con el fin de obtener resultados óptimos, o como decimos, al menos cercanos a los exhibidos por los otros modelos.

Por último, es importante destacar otro camino de progreso descubierto por el trabajo, pues gracias a la dedicación puesta en él, se ha podido comprender la necesidad de poder comparar de forma ágil y cómoda diferentes alternativas dentro de campos tan cambiantes hoy día como el de la visión artificial, el aprendizaje automático o las redes convolucionales. Al respecto, una solución factible pudiese ser el desarrollo de un software que permitiese evaluar el rendimiento de diferentes arquitecturas y datasets, así como distintas configuraciones de hiperparámetros para los modelos.

Podría tratarse de una aplicación simple, que permitiese seleccionar entre un conjunto cerrado de modelos y datasets compatibles según su formato, y habilitase una forma sencilla de modificar los hiperparámetros más importantes que afectan al entrenamiento de una red, como la tasa de aprendizaje, la función de activación, el tamaño de batch, el número de épocas, si aplicar o no dropout, con qué valores y en qué partes de la arquitectura, etc.

Con esto, se conseguiría una herramienta sencilla pero potente, que hiciese posible la realización de comparativas dinámicas, de manera rápida, pudiendo conectar incluso ciertos servicios en la nube, como el utilizado Google Colaboratory, para que se ejecutase ahí el entrenamiento completo de la red, y la aplicación pasase a ser un mero panel de control para preparar y gestionar todos estos procesos.

9. Bibliografía

Ardi, M. (2021, abril 6). *CIFAR-10 Image Classification*. Medium. https://becominghuman.ai/cifar-10-image-classification-fd2ace47c5e8

Basavarajaiah, M. (2019, abril 2). *6 basic things to know about Convolution*. Medium. https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

Bozinovski, S. (2020). Reminder of the First Paper on Transfer Learning in Neural Networks, 1976. *Informatica*, *44*(3), Article 3. https://doi.org/10.31449/inf.v44i3.2828

Bozinovski, S., & Fulgosi, A. (1976). *The influence of pattern similarity and transfer of learning upon training of a base perceptron B2*.

Brian. (2018). *Bringing it On During DIUx xView 2018 Detection Challenge*. https://www.wovenware.com/blog/2018/10/bringing-it-on-during-diux-xview-2018-detection-challenge/

Brownlee, J. (2019, mayo 7). How to Develop a CNN for MNIST Handwritten Digit Classification. *Machine Learning Mastery*. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/

Cao, X., Wipf, D., Wen, F., Duan, G., & Sun, J. (2013). *A Practical Transfer Learning Algorithm for Face Verification*. 3208-3215. https://openaccess.thecvf.com/content_iccv_2013/html/Cao_A_Practical_Transfer_2013_ICCV_paper.html

Captain, S., Captain, S., & Captain, S. (2018, julio 31). *Standard Cognition is first Amazon Go rival to unveil deal with stores*. Fast Company. https://www.fastcompany.com/90211118/standard-cognition-is-first-amazon-go-rival-to-announce-a-deal-with-stores

Cogito. (2019, octubre 16). How Computer Vision Can Improve Accuracy of Diagnosis in Medical Imaging Analysis? *Cogito*. https://www.cogitotech.com/blog/how-computer-vision-can-improve-accuracy-of-diagnosis-in-medical-imaging-analysis/

Cohen, J. (2021, abril 9). *Computer Vision at Tesla*. Medium. https://heartbeat.fritz.ai/computer-vision-at-tesla-cd5e88074376

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*(4), 303-314. https://doi.org/10.1007/BF02551274

Dabydeen, A. (2021). *Data2040_final* [Jupyter Notebook]. project_2. https://github.com/frlim/data2040_final (Original work published 2019)

Deng, L., Hinton, G., & Kingsbury, B. (2013). *New types of deep neural network learning for speech recognition and related applications: An overview*. https://ieeexplore.ieee.org/abstract/document/6639344

DIUx. (2018). *XView*. http://xviewdataset.org/

El Ghawalby, H. (2015). *The Columbia Object Image Library (COIL-100).* ResearchGate. https://www.researchgate.net/figure/The-Columbia-Object-Image-Library-COIL-100_fig1_281978933

Fayek, H. M., Lech, M., & Cavedon, L. (2017). Evaluating deep learning architectures for Speech Emotion Recognition. *Neural Networks*, *92*, 60-68. https://doi.org/10.1016/j.neunet.2017.02.013

Fei-Fei Li & Justin Johnson & Serena Yeung. (2017). *Lecture 11: Detection and Segmentation*. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Felzenszwalb, P., McAllester, D., & Ramanan, D. (2008). A discriminatively trained, multiscale, deformable part model. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 1-8. https://doi.org/10.1109/CVPR.2008.4587597

Freund, Y., & Schapire, R. E. (1999). *A Short Introduction to Boosting*. 14.

Fukushima, K. (1979). Neural network model for a mechanism of pattern recognition unaffected by shift in position-Neocognitron. *IEICE Technical Report, A, 62(10), 658-665*.

Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). *Image Style Transfer Using Convolutional Neural Networks*. 2414-2423. https://openaccess.thecvf.com/content_cvpr_2016/html/Gatys_Image_Style_Transfer_CVPR_2016_paper.html

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*. http://arxiv.org/abs/1406.2661

Gopalakrishnan, K., Khaitan, S. K., Choudhary, A., & Agrawal, A. (2017). Deep Convolutional Neural Networks with transfer learning for computer vision-based data-driven pavement distress detection. *Construction and Building Materials*, *157*, 322-330. https://doi.org/10.1016/j.conbuildmat.2017.09.110

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*. http://arxiv.org/abs/1512.03385

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory*. Psychology Press.

Hu, J., Shen, L., & Sun, G. (2018). *Squeeze-and-Excitation Networks*. 7132-7141. http://openaccess.thecvf.com/content_cvpr_2018/html/Hu_Squeeze-and-Excitation_Networks_CVPR_2018_paper.html

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, *148*(3), 574-591.

Kirsch, R. A., Cahn, L., Ray, C., & Urban, G. H. (1957). Experiments in processing pictorial information with a digital computer. *Papers and discussions presented at the December 9-13, 1957, eastern joint computer conference: Computers with deadlines to meet*, 221-229. https://doi.org/10.1145/1457720.1457763

Korkmaz, S. (2019). *FIGURE 1: A general architecture and workflow of a deep neural networks...* ResearchGate. https://www.researchgate.net/figure/A-general-architecture-and-workflow-of-a-deep-neural-networks-model_fig1_332292257

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. En F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 25* (pp. 1097-1105). Curran Associates, Inc. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Kumar, A. (2020). *Object-recognition-CIFAR-10* [Python]. https://github.com/abhijeet3922/Object-recognition-CIFAR-10 (Original work published 2018)

Kurita, T. (2017). *Fig. 2. The graph of the activation functions ReLU and ELU. ReLU is...* ResearchGate. https://www.researchgate.net/figure/The-graph-of-the-activation-functions-ReLU-and-ELU-ReLU-is-shown-by-the-red-line-and-ELU_fig2_318327308

LaLonde, R., & Bagci, U. (2018). Capsules for Object Segmentation. *arXiv:1804.04241 [cs, stat]*. http://arxiv.org/abs/1804.04241

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541-551. https://doi.org/10.1162/neco.1989.1.4.541

LeCun, Y., Cortes, C., & Burges, C. (1999). *MNIST handwritten digit database*. http://yann.lecun.com/exdb/mnist/

Leung, K. (2021, agosto 24). *The Dying ReLU Problem, Clearly Explained*. Medium. https://towardsdatascience.com/the-dying-relu-problem-clearly-explained-42d0c54e0d24

Linnainmaa, S. (1970). The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, *2*, 1150-1157 vol.2. https://doi.org/10.1109/ICCV.1999.790410

Marr, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information*. Henry Holt and Co. Inc.

MC.AI, MC. A. (2018, mayo 30). Deep learning chapter 1 summary. *Mc.Ai*. https://mc.ai/deep-learning-chapter-1-summary/

Minaee, S. (2019, octubre 28). *20 Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics*. Medium. https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce

Minsky, M. (1961). Steps toward Artificial Intelligence. *Proceedings of the IRE*, *49*(1), 8-30. https://doi.org/10.1109/JRPROC.1961.287775

Minsky, M., & Papert, S. A. (1969). *Perceptrons—An introduction to computational geometry*. The MIT Press. https://mitpress.mit.edu/books/perceptrons

Mitchell, M., Dodge, J., Goyal, A., Yamaguchi, K., Stratos, K., Han, X., Mensch, A., Berg, A., Berg, T., & Daumé III, H. (2012). Midge: Generating Image Descriptions From Computer Vision Detections. *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 747-756. https://www.aclweb.org/anthology/E12-1076

NVIDIA Tesla K80 Specs. (2014). TechPowerUp. https://www.techpowerup.com/gpu-specs/tesla-k80.c2616

NVIDIA Tesla P100 PCIe 16 GB Specs. (2016). TechPowerUp. https://www.techpowerup.com/gpu-specs/tesla-p100-pcie-16-gb.c2888

O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks. *arXiv:1511.08458 [cs]*. http://arxiv.org/abs/1511.08458

Papert, S. A. (1966). *The Summer Vision Project*. https://dspace.mit.edu/handle/1721.1/6125

Park, J., & Sandberg, I. W. (1991). Universal Approximation Using Radial-Basis-Function Networks. *Neural Computation*, *3*(2), 246-257. https://doi.org/10.1162/neco.1991.3.2.246

Paul, S. (2021). *Transfer Learning using a pre-trained VGG16 model on CIFAR-10 dataset.* [Jupyter Notebook]. https://github.com/sayakpaul/Transfer-Learning-with-CIFAR10 (Original work published 2018)

Perez, S. (2017). Target is adding Pinterest's visual search tool to its app and website. *TechCrunch*. https://social.techcrunch.com/2017/09/25/target-is-adding-pinterests-visual-search-tool-to-its-app-and-website/

Qiao, T., Zhang, J., Xu, D., & Tao, D. (2019). *MirrorGAN: Learning Text-To-Image Generation by Redescription*. 1505-1514. https://openaccess.thecvf.com/content_CVPR_2019/html/Qiao_MirrorGAN_Learning_Text-To-Image_Generation_by_Redescription_CVPR_2019_paper.html

Radečić, D. (2020, mayo 20). *Colab Pro: Is it Worth the Money?* Medium. https://towardsdatascience.com/colab-pro-is-it-worth-the-money-32a1744f42a8

Rasul, K. (2017). *Repositorio GitHub Fashion MNIST*. GitHub. https://github.com/zalandoresearch/fashion-mnist

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. 779-788. https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html

Roberts, L. (1963). *Machine Perception of Three-Dimensional Solids*.

Rosenblatt, F. (1957). *The Perceptron—A Perceiving and Recognizing Automaton*. https://blogs.umass.edu/brain-wars/1957-the-birth-of-cognitive-science/the-perceptron-a-perceiving-and-recognizing-automaton/

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533-536. https://doi.org/10.1038/323533a0

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *arXiv:1409.0575 [cs]*. http://arxiv.org/abs/1409.0575

Russakovsky, O., Ma, S., Krause, J., Deng, J., Berg, A., & Li, F.-F. (2014). *ILSVRC2014*. http://www.image-net.org/challenges/LSVRC/2014/ILSVRC2012_val_00004465.png

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, *3*(3), 210-229. https://doi.org/10.1147/rd.33.0210

San Biagio, M. (2014). *Fig. 9. Example images from Pascal VOC 2007 dataset.* ResearchGate. https://www.researchgate.net/figure/Example-images-from-Pascal-VOC-2007-dataset_fig7_271834295

Sandu, R., & Karim, S. (2020). *The application of fast CapsNet computer vision in detecting Covid-19*. *5*, 29-34.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197-227. https://doi.org/10.1007/BF00116037

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*. http://arxiv.org/abs/1409.1556

Stackoverflow, C. (2013). *standard deviation—Does it make sense for non-negative data to subtract the mean and divide by the std dev?* Cross Validated.

https://stats.stackexchange.com/questions/47752/does-it-make-sense-for-non-negative-data-to-subtract-the-mean-and-divide-by-the

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*. http://arxiv.org/abs/1409.4842

Tardi, C. (2021). *Moore's Law Explained*. Investopedia. https://www.investopedia.com/terms/m/mooreslaw.asp

Team, K. (2021). *Keras documentation: Keras Applications*. https://keras.io/api/applications/

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *LIX*(236), 433-460. https://doi.org/10.1093/mind/LIX.236.433

Universidad Internacional de La Rioja, U. (2019). *Tema 3, Asignatura de Aprendizaje Automático (Máster Universitario en Inteligencia Artificial)*. https://campusvirtual.unir.net/

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, *1*, I-I. https://doi.org/10.1109/CVPR.2001.990517

Weng, S.-K., Kuo, C.-M., & Tu, S.-K. (2006). Video object tracking using adaptive Kalman filter. *Journal of Visual Communication and Image Representation*, *17*(6), 1190-1208. https://doi.org/10.1016/j.jvcir.2006.03.004

Werbos, P. J. (1974). New tools for Prediction and Analysis in the Behavioral Sciences. *Ph. D. dissertation, Harvard University*.

Wiley, V., & Lucas, T. (2018). Computer Vision and Image Processing: A Paper Review. *International Journal of Artificial Intelligence Research*, *2*(1), 29-36. https://doi.org/10.29099/ijair.v2i1.42

Zhang, R., Isola, P., & Efros, A. A. (2016). Colorful Image Colorization. En B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer Vision – ECCV 2016* (pp. 649-666). Springer International Publishing. https://doi.org/10.1007/978-3-319-46487-9_40

Anexos

Anexo I. Artículo de investigación

Comparativa de redes convolucionales para clasificación de imágenes en visión artificial

Alberto Bausá Cano

Universidad Internacional de la Rioja, Logroño (España)

Fecha Septiembre 2021

Resumen

El aprendizaje automático y las redes neuronales, con la visión artificial, han sido algunos de los principales campos de investigación de los últimos años dentro de la IA. Además, la incorporación de estas redes a tareas de visión artificial ha propiciado una creciente proliferación de alternativas en arquitecturas, datasets y librerías.

Por ello, se plantea una comparativa que analice las similitudes y diferencias entre tres redes convolucionales, dos de ellas preentrenadas, evaluando su desempeño sobre un mismo dataset, y recolectando métricas que permitan realizar una interpretación de los datos y lograr conclusiones relevantes que puedan ser reaprovechadas en proyectos similares.

Una vez finalizado el estudio, se han logrado algunas pistas interesantes acerca de la utilidad de aplicar transfer learning para abordar la solución de un problema de clasificación de imágenes, comprobando que se debe seleccionar cuidadosamente qué modelo base elegir, hasta qué nivel conservar dicha arquitectura, y cuánto entrenar cada capa.

Palabras Clave

Aprendizaje profundo,
Visión artificial,
Redes convolucionales,
Clasificación de imágenes,
Análisis comparativo

I. Introducción

Hoy día, la Visión Artificial es uno de los campos más potentes y con mayor proyección dentro del ámbito de la Inteligencia Artificial, con multitud de aplicaciones en el mundo real. Enlazado a ese campo, tenemos otro igual de importante: el del Aprendizaje Profundo y las redes neuronales. Durante la última década se ha popularizado un tipo de redes conocidas como convolucionales, especializadas en tareas de Visión Artificial.

Se pueden encontrar sus orígenes históricos en [1] Fukushima (1979), con su Neocognitron para el reconocimiento de patrones visuales en imágenes. Más tarde, sobre ese modelo, el equipo de [2] LeCun et al. (1989) conseguiría aplicar con éxito un proceso de retropropagación para el entrenamiento de la red, dando origen a la primera CNN moderna, LeNet-5. Entre otros logros, también confeccionaron el pionero dataset de imágenes MNIST.

Retomando la última década, se ha producido un apogeo de arquitecturas convolucionales (AlexNet, GoogLeNet, VGGNet, ResNet, etc), así como una proliferación de datasets (Fashion MNIST, CIFAR 10&100, ImageNet…). Además, se han desarrollado multitud de herramientas que facilitaban todo el proceso de principio a fin, como Tensorflow, a día de hoy una de las plataformas más utilizadas en todo el mundo cuando se trata de implementar soluciones basadas en redes profundas.

Debido a estos factores, se logra comprender la magnitud y relevancia que tienen actualmente las redes neuronales convolucionales aplicadas a tareas de visión artificial, como la clasificación de imágenes. Tal diversidad dificulta seleccionar correctamente las más adecuadas, lo que hace necesaria la existencia de estudios previos que permitan obtener cierto conocimiento o experiencia en el campo, con el fin de realizar elecciones más conscientes e informadas.

Por tales motivos, se plantea el desarrollo de una comparativa entre tres arquitecturas de redes convolucionales y un dataset objetivo sobre el que evaluar su rendimiento. Para ajustar más el foco, la intención es centrar el estudio en redes convolucionales especializadas en clasificar imágenes, tarea central de muchos datasets y algunas competiciones como la ILSVRC. Se pretende también comparar diversas técnicas y formas de entrenamiento, distinguiendo así entre una red implementada y entrenada completamente desde cero, hasta la reutilización de un par de modelos preentrenados, mediante *transfer learning*.

Se comenzará haciendo una selección de los modelos, datasets y librerías más apropiadas, para continuar con la búsqueda de las mejores implementaciones para esas alternativas seleccionadas. Después de construir las soluciones propuestas, se ejecutará cada modelo con distintas configuraciones de hiperparámetros, para recolectar una serie de métricas: *accuracy*, *loss*, tiempo de entrenamiento, e impacto en consumo de memoria o disco.

Estas métricas servirán como herramienta para un análisis acerca del rendimiento exhibido por cada modelo y los resultados obtenidos, lo cual nos permitirá discutir las ventajas y desventajas de cada red, extrayendo conclusiones relevantes acerca de los comportamientos observados. Este último paso es fundamental, pues se esperan conseguir contribuciones con posibilidad de utilizarse en futuros proyectos similares.

II. Estado del Arte

Aquí se detalla y documenta el contexto científico del presente artículo, que sirve como base conceptual e histórica. Se comienza con algunas definiciones de conceptos como Visión Artificial, Aprendizaje Automático o Aprendizaje Profundo. También se expone cómo este último tiene recientes aplicaciones en el terreno de la visión artificial, mediante nuevas técnicas como las Redes Neuronales Convolucionales o las Redes Generativas Antagónicas.

A partir de ahí, se hace un repaso histórico que permite encuadrar el marco teórico, mencionando las principales contribuciones realizadas por diversos autores e instituciones, sirviendo de trasfondo para contextualizar la problemática específica sobre la que versa el estudio.

La visión artificial

Se puede considerar la visión artificial como aquella disciplina que combina el procesamiento digital de imagen con el reconocimiento de patrones, extrayendo alguna interpretación de la imagen analizada. En el caso de los algoritmos de visión artificial construidos con aprendizaje automático, mediante redes convolucionales, esa interpretación final de la imagen, cualitativa y cuantitativa, permite obtener información que guíe la toma de decisiones ante diversas situaciones.

Algunas de las tareas o problemáticas más comunes actualmente son las que se aprecian en la Figura 1: clasificación, segmentación semántica, localización, varios objetos, etc.

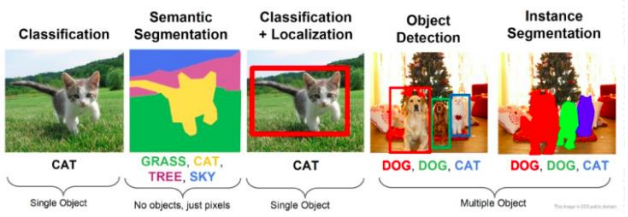

Figura 1. Tareas comunes en visión artificial.
Fuente: [3] Fei-Fei Li & Justin Johnson & Serena Yeung (2017)

En el nuevo milenio, la comunidad científica, impulsada por la necesidad de disponer de métricas estándar de evaluación para pruebas de rendimiento, que permitiesen comparar los diferentes modelos de redes convolucionales, fue creando algunos datasets masivos, como el *Pascal Voc Project* en el año 2005, o *ImageNet* en el año 2010, usado para la competición ILSVRC.

Aprendizaje automático

Se entiende por aprendizaje automático el subcampo, dentro de la Inteligencia Artificial, que se ocupa de investigar y desarrollar técnicas que permitan que las máquinas aprendan. Es decir, que mejoren su rendimiento gracias a experiencia adquirida mediante una fase de entrenamiento previa.

Sus avances más recientes se han debido a dos motivos:

- Por un lado, el uso masivo de Internet provocó un aumento de los datos disponibles. La escasez de los mismos era una de las razones que impedía a las redes neuronales alcanzar todo su potencial, o no conseguir la precisión deseada.
- Por otro lado, la mejora en la capacidad de computación logró reducir el tiempo de entrenamiento, permitiendo modelos complejos para problemas cada vez más difíciles.

Aprendizaje profundo

En IA, entendemos por aprendizaje profundo la disciplina que aglutina los algoritmos y las técnicas que pretenden modelar representaciones de la realidad, usando arquitecturas complejas y jerarquizadas que profundizan hasta diversos niveles de detalle, desde abstracciones más simples hasta otras más complejas. Conocemos estas estructuras como redes neuronales.

Tiempo después surgió un tipo especial de esta clase de redes, conocidas como convolucionales, y especializadas en el procesamiento de imágenes. Estas se caracterizan por disponer de un nuevo tipo de capa que aplica operaciones de convolución sobre las imágenes de entrada, mediante filtros, obteniendo mapas de activación que ayudan a extraer características útiles.

Como apunte breve, se menciona también la técnica de *transfer learning*, por ser empleada a lo largo del estudio. Se conoce como tal ([4] Bozinovski & Fulgosi, 1976) a una técnica y a un campo de investigación propio dentro del aprendizaje automático, centrado en estudiar el cómo almacenar y aplicar el conocimiento adquirido por un primer modelo o red, sobre un primer problema de aprendizaje, para utilizarlo más tarde como punto de partida con un segundo modelo, sobre un segundo problema de aprendizaje diferente, pero relacionado, logrando aprovechar el entrenamiento ya realizado.

Redes neuronales para visión artificial

Una vez realizado un repaso breve por las aportaciones dentro de los campos del aprendizaje automático y la visión artificial, procede mencionar el intento de aplicar las redes neuronales a problemas de este último, consiguiendo mejorar muchos de los resultados obtenidos anteriormente con otras técnicas.

Estas redes han conseguido resolver de manera eficiente muchas de las tareas principales del campo de la visión artificial, como la clasificación de imágenes, la segmentación de objetos, el seguimiento de objetos en video, la transferencia de estilo en imágenes o la generación de imágenes a partir de texto. Dichas soluciones, a su vez, han posibilitado el desarrollo de muchos sectores, como el cuidado de la salud, la conducción autónoma, el comercio desatendido o el marketing mejorado.

Tales consecuencias tienen su origen en diversas arquitecturas de redes convolucionales surgidas durante la última década, las cuales mejoraron año a año los resultados de sus predecesoras. Muchas de ellas se habían dado a conocer en competiciones como la ILSVRC, siendo las ganadoras de algunas de las ediciones, como son los casos de AlexNet, GoogLeNet, VGGNet o ResNet.

Marco tecnológico actual

Se realiza en este apartado un rápido repaso por el panorama tecnológico actual en lo referente al aprendizaje automático, como una ampliación del estado del arte a un nivel más técnico. En dicho panorama, podemos encontrar diferentes librerías, frameworks o datasets, la mayoría de muy reciente aparición, que pujan por ser las soluciones más utilizadas hoy en día.

Todas ellas, especialmente las centradas en redes neuronales, presentan notables ventajas: simplifican la construcción de modelos complejos, permiten automatizar ciertos pasos del proceso, facilitan la gestión del entrenamiento y aumentan la eficiencia de las ejecuciones, pues utilizan implementaciones altamente optimizadas para sus algoritmos y estructuras de datos.

Como ejemplos de librerías y frameworks, encontramos Tensorflow, Keras, PyTorch, Microsoft CNTK o sci-kit learn. En el caso de los datasets, tenemos MNIST, CIFAR 10&100, COIL-100 o el masivo y público Open Images de Google.

Recapitulación

Después de este resumen de algunas de las aportaciones en el campo de las redes convolucionales, podemos llegar a una conclusión: año tras año surgen más arquitecturas, generalmente de mayor tamaño y complejidad, y nuevas combinaciones sobre

algunas de ellas con el fin de mejorar los resultados conseguidos.

Debido a esta enorme variedad de alternativas, se vuelve transcendental la existencia de estudios comparativos entras ellas, para que tanto investigadores como profesionales dispongan de análisis detallados del rendimiento de unas y otras. De ahí nace la idea y la oportunidad del presente trabajo.

III. Objetivos y Metodología

Considerando lo visto anteriormente, se concluye que el objetivo principal de este trabajo es **realizar un análisis comparativo que evalúe el rendimiento ofrecido por distintas arquitecturas de redes neuronales centradas en tareas de clasificación de imágenes**. Con el fin de alcanzar dicho objetivo general, se han planteado los siguientes objetivos específicos:

1. Seleccionar las alternativas más adecuadas para utilizar, justificando los motivos de su elección a partir de la revisión del estado del arte realizada previamente.

2. Hallar las mejores implementaciones para las arquitecturas seleccionadas, y las técnicas de preprocesado de datos o mejoras para el entrenamiento del modelo óptimas.

3. Desarrollar el análisis comparativo, ejecutando los modelos elegidos anteriormente y recolectando diferentes métricas

4. Analizar el resultado, interpretándolo y estudiando ventajas y desventajas de cada opción, sacando conclusiones relevantes.

Para lograr el objetivo general y los específicos, se ha propuesto la **metodología** expuesta a continuación:

Paso 1. Proceso de selección de las mejores alternativas, sobre un conjunto de frameworks, arquitecturas y datasets, exponiendo los motivos que explican su idoneidad.

Paso 2. Construcción y adaptación de las diferentes arquitecturas, a partir de los modelos elegidos antes. También se efectúan los ajustes requeridos en cuanto al preprocesamiento del dataset o las técnicas para optimizar la convergencia de la red.

Paso 3. Ejecución de los modelos desarrollados con el fin de recolectar un compendio de métricas para su posterior estudio.

Paso 4. Análisis de los resultados de la etapa anterior, con el propósito de sacar conclusiones importantes que expliquen la utilidad, ventajas y desventajas de cada una de las arquitecturas. Es el paso fundamental del trabajo, además del propio desarrollo de la comparativa realizado.

IV. Contribución

Las principales contribuciones del presente estudio han sido las conclusiones finales, producto del análisis de los resultados, que permiten ayudar a entender mejor cuales de las arquitecturas pueden ser mejores respecto a otras, y qué técnicas y métodos pueden optimizar las metas alcanzadas.

En ese sentido, se han extraído algunas ideas interesantes acerca de la técnica conocida como *transfer learning*, viendo que no es suficiente por sí misma para justificar su uso en cualquier escenario o como aproximación para cualquier solución. Se ha demostrado como una herramienta útil cuando se conoce bien el problema a resolver, y se realizan las adaptaciones necesarias al mismo, eligiendo minuciosamente qué modelos preentrenados utilizar como base de nuestra arquitectura, cuánto conservar de dichos modelos, o hasta qué nivel entrenar cada capa.

En el caso de que no se aplique de forma cuidadosa, no dotando de suficiente profundidad a la red final, o evitando que se entrenen capas que sí deberían hacerlo, se pueden obtener resultados no deseados que planteen la necesidad de reconstruir la arquitectura desde un principio, o incluso, reevaluar si los modelos elegidos son los adecuados para la tarea a solventar.

V. Evaluación

Sucintamente, se relata en esta sección la explicación de las soluciones a comparar y se enumeran las métricas que conforman los resultados. Para poder evaluar las arquitecturas seleccionadas se ha elegido uno de los datasets más utilizados, *cifar10*, siendo los criterios para su elección los siguientes:

- Está enfocado en la clasificación de imágenes.
- Tiene un tamaño medio, lo que lo hace cómodo y manejable, además de suficientemente complejo como para obtener conclusiones destacadas. Es un término medio adecuado.
- Es distinto a ImageNet, el usado por las redes preentrenadas de la comparativa. Así, se dispone de un conjunto de datos nuevo sobre el que evaluar la utilidad del *transfer learning*.

Custom CNN

Para este modelo se ha optado por una arquitectura extraída de [5] Kumar (2018/2020) y adaptada para los propósitos del estudio. Su elección se ha basado en los buenos resultados demostrados (~90% de *accuracy*), así como en el uso que hace de varias técnicas bastante útiles para el aprendizaje de la red.

Cuenta con seis capas convolucionales y 309.290 parámetros, lo que la convierte en una estructura relativamente sencilla. Está conformada por la repetición de un patrón principal de capas (convolucional, activación y normalización). Podemos encontrar un patrón secundario (max pooling y dropout), después de alguno de los bloques principales. Finaliza con una capa *flatten*, que conecta con la salida de 10 neuronas, con activación *softmax*.

Teniendo esto en cuenta, se describe su ejecución durante el entrenamiento. Primeramente, se carga el dataset y se divide en un conjunto de entrenamiento (50 mil ejemplos) y otro de validación (10 mil), usado también para test. A tales conjuntos se les aplica una normalización *z-score* ([6] Cross Validated (2013)). Posteriormente, se codifican los vectores de etiquetas en formato *one-hot* y se aplica *data augmentation*, para ampliar el conjunto de datos, enriqueciendo la información para el entrenamiento.

Finalmente se entrena el modelo, usando tamaño de batch con 64 elementos, optimizador *RMSprop* y *categorical crossentropy* como función de pérdida. Además, se usa una tasa de aprendizaje adaptativa: 0.001 (épocas 1-75), 0.0005 (76-100), y 0.0003 (101 en adelante). Con una tasa más alta en los primeros momentos de mayor incertidumbre, aumentando la velocidad de convergencia, que se ve reducida progresivamente, para que la red aprenda más lentamente y de forma generalmente más fiable.

VGGNet 16

La arquitectura original *vgg16* cuenta con 138 millones de parámetros y una implementación de 16 capas convolucionales (**C**) y densas (**D**), y 5 max. pooling (**P**), ordenadas de esta forma:

CCP – **CCP** – **CCCP** – **CCCP** – **CCCP** – **DDD**

Con el fin de adaptarlo a este trabajo, el modelo inicial ha sido modificado, gracias al guion de [7] Paul (2018/2021). Conserva los 3 primeros bloques, a partir de los cuales se ha añadido una salida compuesta por una capa de pooling, una de normalización, y dos capas densas (256 neuronas), seguidas por una dropout y otra densa (10 neuronas) *softmax*. Así, el número de parámetros desciende hasta los 1.8 millones, y tan solo 134 mil entrenables, que son los correspondientes a las nuevas capas.

En cuanto al entrenamiento, se carga el dataset y se divide en conjunto de entrenamiento (50 mil) y test (10 mil). Por su parte, el primero reserva el 15% del total para un conjunto de validación independiente, y asigna el 85% restante para el entrenamiento, terminando así con tres conjuntos de imágenes. Para cada grupo, sus vectores de etiquetas se codifican como *one-hot*, y se aplica

una redimensión de sus imágenes, hasta el tamaño 48x48x3 píxeles, compatible con *vgg16*. A continuación, se aplica *data augmentation* para el conjunto de entrenamiento y el de validación, en sendos casos con las mismas transformaciones. Por último, se compila el modelo utilizando *Adam* (*learning rate* = 0.0004) y *binary crossentropy* como función de coste.

Se ejecuta el entrenamiento con 32 elementos por batch, aplicando una tasa de aprendizaje diferencial, dado que las capas congeladas no son entrenadas (*learning rate* = 0). El resto de la arquitectura tiene el *learning rate* comentado antes, 0.0004.

ResNet 50

Existen diferencias respecto al modelo original, con ideas extraídas de [8] Dabydeen (2019). Al principio se introducen 3 capas para reescalar cada imagen hasta 256x256x3 píxeles, lo admitido por ImageNet. Seguidamente, los datos llegan al modelo base *resnet50*, se aplana su salida y continua por dos capas densas (128 y 64 neuronas), con normalización y dropout del 50%. Cuenta con una capa densa final (10 neuronas *softmax*). Destacar que en este caso no se ha congelado ninguna capa, reentrenando la red al completo. Partiendo de los 23 millones de parámetros del modelo base, se alcanzan los 40 millones, todos entrenables.

En relación al preprocesamiento de datos, parte se realiza en la propia red, con las capas de reescalado que aumentan cada imagen. Además, se divide el conjunto de datos inicial en dos (entrenamiento y validación/test). Ambos son normalizados, así como se codifican en formato *one-hot* los vectores de salida de la red. Posteriormente, se compila el modelo con un optimizador *RMSprop* (*learning rate* = 0.00002) junto con la función de coste *binary crossentropy*, y un tamaño de 20 elementos por batch.

Métricas empleadas

Por último, es momento de enumerar las distintas métricas recolectadas como consecuencia de las diversas ejecuciones de los modelos, las cuales ofrecen un camino para inferir similitudes y diferencias entre las arquitecturas utilizadas. Contamos con: *accuracy (*entrenamiento, validación, test), *loss (*entrenamiento, validación, test), tiempo de ejecución y número de épocas, así como consumo de memoria y espacio en disco ocupado.

VI. RESULTADOS

El **modelo *custom-cnn***, con el proceso de entrenamiento explicado antes, ha conseguido los resultados de la Figura 2:

Métrica / Época	#60	#100	#150	#300
Tiempo	24m 19s	39m 5s	58m 45s	2h 3m 4s
Accuracy	0.8498	0.8823	0.8877	0.8967
Loss	0.5905	0.4524	0.4342	0.4019
Val. / Test Accuracy	0.8575	0.8701	0.8871	0.8768
Val. / Test Loss	0.5876	0.5188	0.4707	0.4909

Figura 2. Resultados de entrenamiento del modelo *custom-cnn*.
Fuente: elaboración propia

Como ocurre usualmente, la *accuracy* más alta (89.67%) se registra para el mayor número de épocas, con 300 ejecutadas en algo más de 2 horas. La *loss* es inversamente proporcional. No se presentan individualmente los valores para test pues se usa el mismo conjunto que para validación, coincidiendo los valores.

Respecto al consumo de memoria o espacio en disco, el dataset apenas alcanza los 150 MB, y el modelo junto con sus pesos no llegan a los 2 MB. Sin embargo, si se comprobó una ocupación de 10/16 GB en relación a la memoria de video de la GPU.

Para el modelo de **red *vgg16***, con los parámetros especificados anteriormente, nos encontramos con los resultados de la Figura 3:

Métrica / Época	#40	#100	#200	#500
Tiempo	6m 47s	16m 50s	33m 44s	1h 25m 44s
Accuracy	0.6861	0.7658	0.8282	0.8908
Loss	0.1494	0.1172	0.0895	0.0600
Val. Accuracy	0.6740	0.6907	0.6867	0.6796
Val. Loss	0.1522	0.1498	0.1706	0.2307

Figura 3. Resultados de entrenamiento del modelo *vgg16*.
Fuente: elaboración propia

Tenemos como mejor resultado un 89.08% en *accuracy*, tras 500 épocas ejecutadas y casi una hora y media. Los valores para *test accuracy* / *test loss* serían: 12.06% / 258.37 (#40), 14.14% / 568.17 (#100), 9.03% / 1439.07 (#200) y 9.93% / 3588.07 (#500). En otra prueba realizada por separado, que utilizaba dos conjuntos (entrenamiento y validación/test), se obtuvieron incluso peores resultados para validación, siendo estos iguales a los vistos en los valores de test de la ejecución anterior (con tres grupos de datos).

Por último, comentar que el modelo y pesos de la red ocupan solo unos 8 MB, pero el consumo de memoria de la GPU durante el entrenamiento fue elevado, con 7-8 GB ocupados de media.

El **modelo *resnet50*** ha alcanzado el resultado de la Figura 4.

Métrica / Época	#5	#10	#20
Tiempo	1h 1m 24s	2h 2m 6s	3h 54m 2s
Accuracy	0.8572	0.9597	0.9815
Loss	0.2352	0.0808	0.0317
Val. / Test Accuracy	0.9291	0.9501	0.9442
Val. / Test Loss	0.1222	0.0419	0.0415

Figura 4. Resultados de entrenamiento del modelo *resnet50*.
Fuente: elaboración propia

Se debe prestar especial atención al valor de *accuracy* logrado, 98.15%, el mejor entre las tres arquitecturas, tras 20 épocas y casi cuatro horas de ejecución, el cual se ve correspondido por una elevada *validation/test accuracy* del 94.42%. Como se ha explicado, se usa el mismo conjunto de datos para validación y test, por lo que no es necesario repetir estos últimos por separado.

Como colación, se resalta el importante tamaño que ocupan el modelo y los pesos, con 156 MB, debido a la enorme cantidad de parámetros entrenables que maneja. Una vez más, el consumo de memoria de video se situó en 10-12 GB ocupados.

VII. DISCUSIÓN

Se discutirán aquí los resultados obtenidos de forma previa, intentando obtener conclusiones e interpretarlos mediante un análisis de los mismos. Haciendo un repaso por el tamaño de cada modelo, medido en parámetros entrenables, tendríamos: *custom-cnn* (310 mil), *vgg16* (135 mil) y *resnet50* (40 millones).

Mientras que el modelo base de *vgg16* tenía 138 millones de parámetros, solo se ha reutilizado parcialmente (~1.7 millones) y además se ha congelado, conservando únicamente 135 mil parámetros entrenables, los de las capas densas añadidas después. Por otro lado, para el modelo *resnet50* se ha reutilizado la arquitectura original por completo (excepto la última capa densa de salida con 1000 neuronas) y se han añadido unas capas densas finales, con 17 millones de parámetros extra, para alcanzar los 40.

Con este breve apunte es posible concebir el poder expresivo y de aprendizaje que puede llegar a alcanzar cada modelo, pues comúnmente, a mayor número de parámetros entrenables mayor

capacidad tendrá la red para generalizar de forma apropiada y realizar predicciones ante ejemplos no vistos anteriormente.

A continuación, se realizará un análisis basado en dos gráficos globales con la *accuracy* y *loss* de cada red, diferenciando cada modelo con una gama cromática distinta. Son las Figuras 5 y 6.

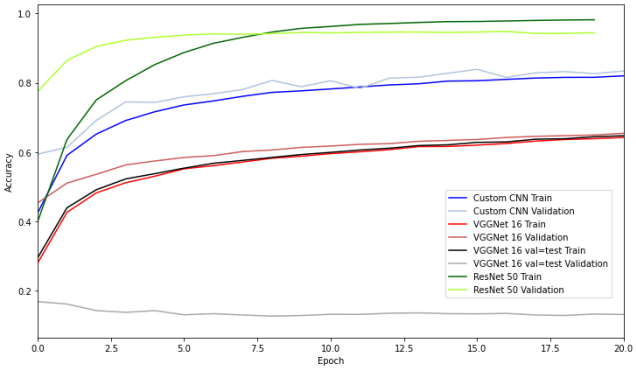

Figura 5. Gráfica con la *accuracy* global (época 0-20).
Fuente: elaboración propia

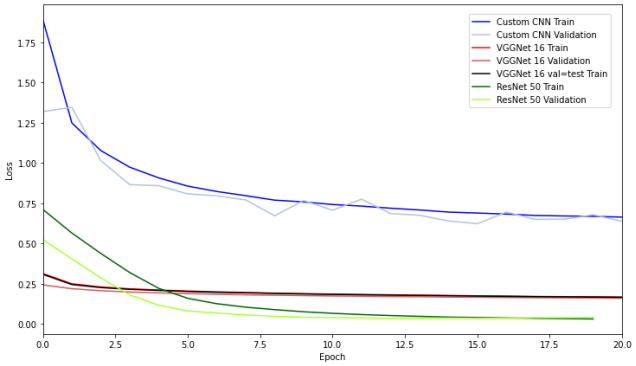

Figura 6. Gráfica con la *loss* global (época 0-20).
Fuente: elaboración propia

Comenzamos hablando de la Figura 5, en la cual podemos ver en último lugar los modelos *vgg16* (dos y tres conjuntos de datos). Sus curvas de *training accuracy* son bastante similares, aunque se diferencian en sus curvas de validación, pues el modelo con tres conjuntos obtiene mejores resultados, pero pronto alcanza un estancamiento en el 70% de *validation accuracy*. Vemos resultados muy inferiores para la ejecución con dos conjuntos.

Les sigue *custom-cnn*, con el segundo mejor desempeño (89.67%) y unas curvas de aprendizaje coherentes entre entrenamiento y validación. En primer lugar destaca visiblemente la arquitectura *resnet50*, con la mejor marca (98.15%).

Se perciben ideas similares cuando nos fijamos en la Figura 6, para *loss*, con *resnet50* una vez más consiguiendo la mayor reducción en su función de coste, mientras que, curiosamente, en este caso es la arquitectura *custom-cnn* la que menos logra decrementar su pérdida. Así, se pone de manifiesto que se pueden manejar valores de coste más altos que otra arquitectura, dentro de unos límites, y aun así obtener mejor *accuracy*.

Con toda esta información, podemos concluir que ambas aproximaciones, tanto la de implementar y entrenar modelos completamente desde cero, como la de reutilizar redes preentrenadas, de forma total o parcial, son válidas de cara a conseguir unos objetivos aceptables sobre CIFAR 10.

Resalta, de manera clara que no siempre el hecho de utilizar redes preentrenadas y aplicar *transfer learning* es suficiente para alcanzar buenas metas, sino que ese proceso se debe realizar con sumo cuidado, adaptando las particularidades de cada problema. Se debe considerar detenidamente qué partes del modelo base se quieren conservar, cuáles no, así como también decidir acerca de qué porcentaje de las partes reutilizadas será o no será entrenable.

Por otro lado, hay que considerar que la técnica de *transfer learning* es capaz de ofrecer mejores resultados cuando el nuevo problema al que se enfrenta la red preentrenada, aunque distinto, tenga similitudes con el problema sobre el que se entrenó originalmente. De lo contrario, el preentrenamiento podría carecer de utilidad, al no disponer la red de un aprendizaje suficiente en el nuevo ámbito en el que se ve envuelta. Así pues, se debe intentar lograr un equilibrio entre los resultados ofrecidos por un modelo, y la facilidad de construirlo.

VIII. Conclusiones

A lo largo del presente artículo se ha realizado un análisis comparativo de diferentes arquitecturas de redes neuronales convolucionales, las cuales copan desde hace unos años el estado del arte en cuanto al nivel de rendimiento que puede llegar a ofrecer cualquier otra técnica en la visión artificial. El propósito ha sido el de encontrar qué modelos permiten alcanzar mejores resultados y qué similitudes y diferencias existen entre ellos, finalizando el estudio con algunas conclusiones relevantes.

Con ese guion en mente, se comienza por un minucioso análisis del estado del arte, repasando las principales arquitecturas de redes convolucionales de la última década. Se ha proseguido recapitulando algunas ideas acerca de la revisión realizada, como que existen multitud de modelos, técnicas y métodos, cada uno de los cuales más o menos apropiado según el caso. Así pues, la selección es en ocasiones compleja, y en muchas situaciones el rendimiento o la calidad alcanzada dependerá de adaptar varias de estas arquitecturas o técnicas al problema concreto.

Lo anterior culminó con la idea de un análisis que comparase y evaluase el rendimiento que ofrecían algunas de las diferentes arquitecturas descubiertas, centrando el foco en la clasificación de imágenes. Se plantearon unos objetivos específicos, los cuales pretendían desgranar el general, entre ellos:

- Se comenzó con un proceso de selección sobre los modelos, datasets y librerías descubiertos gracias al análisis del estado del arte, justificando su elección en cada caso.
- Más tarde, se realizó un estudio más técnico, para elegir qué implementaciones podían ser las óptimas, dentro de las disponibles para cada modelo convolucional seleccionado, y qué técnicas o métodos era conveniente aplicar.
- Una vez adquirido ese conocimiento, se desarrolló la comparativa ejecutando de manera secuencial las diferentes configuraciones de hiperparámetros para cada modelo, y recolectando varias métricas que serían utilizadas luego.
- Por último, se concluyó el estudio con la interpretación de los datos obtenidos como producto de la comparativa, analizando cada caso mediante gráficas que mostrasen el comportamiento para cada modelo y métrica utilizada. Debido a eso se les pudo dar un significado, extrayendo conclusiones relevantes que podrían ser de utilidad para futuros proyectos de similar índole.

Tras estas líneas, se cree satisfecho el objetivo general del presente estudio, pues se ha planteado, desarrollado y debatido acerca del análisis comparativo ideado desde un inicio entre diversas redes centradas en la clasificación de imágenes. Por otro lado, también se consideran cumplidos los objetivos específicos, repasados a lo largo de los últimos párrafos.

Finalmente, se hace necesario comentar que, si bien se consideran superados los objetivos y desarrollada la idea sobre la que se deseaba llevar a cabo la investigación, quedan algunas sombras en el horizonte, en forma de limitaciones encontradas a lo largo del camino, que imposibilitaron un mejor desenlace. Especialmente las relativas a la arquitectura *vgg16*, la cual no se ha conseguido ni siquiera que sea una opción viable, por lo que se deja como una de las tareas pendientes.

Partiendo de la idea del último párrafo, se compartirán de forma escueta algunas reflexiones acerca de qué mejoras se podrían haber realizado en el presente estudio, ya desde un comienzo, o a posteriori, viendo los resultados alcanzados.

Por un lado, la comparativa deja clara la enorme variedad de alternativas dentro de las redes neuronales convolucionales, en el ámbito de las tareas relacionadas con la visión artificial, lo que abre la puerta a escalar el estudio en diferentes direcciones.

Por ejemplo, se podrían enfrentar más grupos diferentes de arquitecturas, acudiendo a otros paradigmas, como el caso de las redes generativas antagónicas (GAN), o la aproximación al campo realizada por [9] Sandu & Karim (2020), con sus redes de cápsulas (*CapsNets)*, las cuales pretenden superar algunas de las limitaciones conocidas de las redes convolucionales. De igual forma, se podría haber añadido algún otro dataset para comprobar las diferencias de comportamiento según el conjunto de datos utilizado para aprender. Entre ellos, CIFAR-100 o COIL-100.

Otra de las posibles mejoras sería la de estudiar detenidamente la arquitectura del modelo *vgg16*, examinando las causas concretas que condujeron a tan mal rendimiento, y planteando las modificaciones oportunas. Por ejemplo, se podría añadir más profundidad a la arquitectura, con más capas convolucionales. Asimismo, sería conveniente explorar diferentes combinaciones de técnicas a aplicar, optimizadores más útiles, o inicializaciones de los parámetros diferentes, con el fin de obtener resultados óptimos, o al menos en consonancia con los otros modelos.

Por último, otra vía de progreso sería el desarrollo de un software que permitiese, de forma ágil y cómoda, evaluar el rendimiento de diferentes arquitecturas y datasets, así como distintas configuraciones de hiperparámetros para los modelos. Podría tratarse de una aplicación que permitiese seleccionar entre un conjunto cerrado de modelos y datasets compatibles según su formato, y habilitase una forma sencilla de modificar los hiperparámetros más importantes que afectan al entrenamiento de la red, como la tasa de aprendizaje, la función de activación, el tamaño de batch, el número de épocas, si aplicar o no dropout, con qué valores y en qué partes de la arquitectura, etc.

Con esto, se conseguiría una herramienta sencilla pero potente, que posibilitase la realización de comparativas dinámicas, de manera rápida, pudiendo conectar incluso ciertos servicios o plataformas en la nube, como Google Colaboratory, utilizado en el presente estudio, para que se ejecutase ahí el entrenamiento completo de la red, y la aplicación fuese un mero panel de control para preparar y gestionar todos estos procesos.

Referencias

[1] Fukushima, K. (1979). Neural network model for a mechanism of pattern recognition unaffected by shift in position-Neocognitron. IEICE Technical Report, A, 62(10), 658-665.

[2] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation, 1(4), 541-551. https://doi.org/10.1162/neco.1989.1.4.541

[3] Fei-Fei Li & Justin Johnson & Serena Yeung. (2017). Lecture 11: Detection and Segmentation. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

[4] Bozinovski, S., & Fulgosi, A. (1976). The influence of pattern similarity and transfer of learning upon training of a base perceptron B2.

[5] Kumar, A. (2020). Object-recognition-CIFAR-10 [Python]. (Original work published 2018) https://github.com/abhijeet3922/Object-recognition-CIFAR-10

[6] Cross Validated (2013). standard deviation—Does it make sense for non-negative data to subtract the mean and divide by the std dev? https://stats.stackexchange.com/questions/47752/does-it-make-sense-for-non-negative-data-to-subtract-the-mean-and-divide-by-the

[7] Paul, S. (2021). Transfer Learning using a pre-trained VGG16 model on CIFAR-10 dataset. (Original work published 2018) [Jupyter Notebook]. https://github.com/sayakpaul/Transfer-Learning-with-CIFAR10

[8] Dabydeen, A. (2021). Data2040_final. project_2. [Jupyter Notebook]. https://github.com/frlim/data2040_final (Original work published 2019)

[9] Sandu, R., & Karim, S. (2020). The application of fast CapsNet computer vision in detecting Covid-19. 5, 29-34.

Anexo II. Listado de acrónimos

AD - Automatic Differentiation (Diferenciación Automática)

CNN - Convolutional Neural Network (Redes Neuronales Convolucionales)

CV - Computer Vision (Visión Artificial)

DL - Deep Learning (Aprendizaje Profundo)

DPM - Deformable Part Model (Modelo de Partes Deformables)

ELU – Exponential Linear Unit (Unidad Lineal Exponencial)

FC - Fully Connected (Completamente Conectado)

FP – Floating Point (Punto Flotante)

GAN - Generative Adversarial Network (Redes Generativas Antagónicas)

GPU - Graphic Processing Unit (Unidad de Procesamiento Gráfica)

ILSVRC - ImageNet Large Scale Visual Recognition Competition (Competición para el Reconocimiento Visual a Gran Escala – de imágenes - de ImageNet)

MIT - Massachusetts Institute of Technology (Instituto Tecnológico de Massachusetts)

ML - Machine Learning (Aprendizaje Automático)

MNIST database - Modified National Institute of Standards and Technology database (Base de datos Modificada del Instituto Nacional de Estándares y Tecnología)

MSE – Mean Square Error (Error Cuadrático Medio)

ReLU - Rectified Linear Unit (Unidad Linear Rectificada)

RL - Representation Learning (Aprendizaje de Representación)

ROI – Region Of Interest (Zona de Interés)

SGD - Stochastic Gradient Descent (Descenso de Gradiente Estocástico)

VOC – Visual Object Classes (Clases de Objetos Visuales)