



Universidad Politécnica  
de Madrid

Escuela Técnica Superior de  
Ingenieros Informáticos



Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

**Redes Neuronales Convolucionales Evolutivas para  
un Caso Práctico de Clasificación de Imágenes**

Autora: Mónica Apellaniz Portos

Tutores: Ángela Ribeiro (Centro de automática y robótica, UPM-CSIC)  
Javier de Lope Asiaín

Madrid, Julio 2022

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

*Trabajo Fin de Máster*  
*Máster Universitario en Inteligencia Artificial*

*Título:* Redes Neuronales Convolucionales Evolutivas para un Caso Práctico de Clasificación de Imágenes

Julio 2022

*Autora:* Mónica Apellaniz Portos

*Tutores:* Ángela Ribeiro (Centro de automática y robótica, UPM-CSIC)  
Javier de Lope Asiain

Inteligencia Artificial  
ETSI Informáticos  
Universidad Politécnica de Madrid

# Agradecimientos

*Gracias a mis tutores Ángela y Javier.*

*Por ayudarme con este proyecto. Por enseñarme a mejorar. Por el apoyo que me han dado durante toda esta etapa y el interés que han mostrado durante esta etapa. Muchas gracias por todo.*

*Gracias a mis compañeros y amigos que han caminado junto a mí en esta aventura.*

*Por el apoyo y comprensión. Por seguir alegrándonos con nuestros logros y sacando sonrisas siempre que las necesitemos.*

*Gracias a mis hermanas, por ese “amor de hermanas” que aunque no se entienda, siempre está. Gracias por nuestro apoyo tan leal e incondicional.*

*Gracias una y mil veces más a mi madre. Por ser, estar y aguantar. Te quiero.*

*Y gracias papá, porque has sido, eres y serás siempre mi fuente inagotable de inspiración.*



# Resumen

En este proyecto se propone un método novedoso y único, basado en procesos evolutivos para el diseño y aplicación de redes convolucionales (CNNs) en problemas de clasificación de imágenes; se buscan redes más pequeñas, simples, y con una precisión igual o superior a la obtenida con otras más complejas ya existentes. Para ello, se diseña un método flexible, capaz de adaptarse a la complejidad del problema a resolver, generando modelos y entrenamientos mucho más eficientes.

El desarrollo evolutivo, sigue una estructura y comportamiento bioinspirado, basado en el algoritmo multiobjetivo y elitista NSGA-II. Se propone un diseño personalizado del algoritmo para este problema concreto, basado en una codificación capaz de representar diversas CNNs con diferentes tamaños, profundidad, distintas conexiones entre capas, etc. También se define una función de evaluación particular, donde la *fitness* se encarga de encontrar una solución acorde a los requisitos o requerimientos del problema, para finalmente conseguir que el algoritmo converja en el óptimo global y así se obtenga la solución deseada.

Este código se ejecuta en remoto, en el centro de supercomputación de Galicia (CESGA), en concreto en el servidor Finisterrae-III. Se sigue un procedimiento progresivo, donde se afrontan numerosos retos y objetivos; se va entendiendo y abordando poco a poco el sistema propuesto, modificándolo hasta obtener una primera aproximación funcional del algoritmo.

Con estos experimentos se consigue ajustar los dos objetivos definidos. Por un lado se minimiza la complejidad de la red, reduciendo el número de parámetros respecto los que tienen otras redes como la ResNet-50 y la Inception-v3, consiguiendo que esta cifra se reduzca hasta en un 99 %, partiendo de redes con 25 y 23 millones de parámetros y con 50 y 159 capas, a redes generadas con el algoritmo propuesto que no llegan ni a los 300000 parámetros, y donde se consiguen redes hasta con 10 o 20 capas. También se consigue maximizar la capacidad de etiquetado aplicada a un problema complejo y no trivial de clasificación de imágenes de malas hierbas, obteniendo hasta un 89 % precisión de clasificación en el proceso de test realizado con imágenes desconocidas para el modelo.



# Abstract

In this project, it is proposed a novel and unique method based on evolutionary processes for the design and application of convolutional networks (CNNs) in image classification problems. The search focuses on finding smaller, simpler networks with a higher or equal classification accuracy than other more complex and already developed networks. For that, a flexible method is designed, able to adapt itself due to the problem complexity, generating much more efficient training and models.

The evolutionary strategy follows a bio-inspired structure and behavior, based on the multi-objective and elitist algorithm NSGA-II. This algorithm is customize for the specific proposed problem, based on a codification capable of representing different CNNs with diverse size, depth, numer layers connections, etc. Also, this methodology presents a particular evaluation function, where the fitness value has to find a solution according to the defined problem requirements, to finally converge on the global optimum, and obtain the desired solution.

This code is remotely executed at the Galician supercomputing center (CESGA), specifically on the Finisterrae-III server. This work follows a progressive procedure where numerous challenges are faced; the proposed system is gradually understood and approached, modifying it until obtaining a first functional algorithm.

With these experiments, it is possible to adjust the two defined objectives. On the one hand, the complexity of the network is minimized, reducing the number of parameters compared to other networks such as ResNet-50 and Inception-v3, reducing this up to 99 %, starting from networks with 25 and 23 million of parameters and 50 and 159 layers, to generated networks with no more than 300,000 parameters and 10 or 20 layers. Also, it is possible to maximize the classification capability with a complex and non-trivial image classification problem, obtaining up to 89 % of accuracy in the test procedure, evaluating these models with unknown images.



# Tabla de contenidos

<b>1. Introducción</b>	<b>I</b>
1.1. Motivación . . . . .	I
1.2. Objetivos . . . . .	2
1.3. Estructura . . . . .	2
<b>2. Estado del arte</b>	<b>3</b>
2.1. Agricultura de Precisión . . . . .	3
2.1.1. Métodos de control de malas hierbas . . . . .	3
2.1.1.1. Técnicas de tratamiento . . . . .	4
2.1.1.2. Técnicas de localización . . . . .	4
2.2. Métodos de Detección: clasificación de imágenes . . . . .	5
2.2.1. Aprendizaje automático . . . . .	5
2.2.1.1. Aprendizaje automático para la detección de malas hierbas . . . . .	6
2.2.2. Computación evolutiva . . . . .	8
2.2.2.1. CE para la detección de malas hierbas . . . . .	8
2.3. Aprendizaje profundo evolutivo . . . . .	8
<b>3. Marco Teórico</b>	<b>II</b>
3.1. Inteligencia Artificial . . . . .	II
3.1.1. Aprendizaje automático . . . . .	12
3.1.2. Aprendizaje Profundo . . . . .	13
3.1.2.1. Red Neuronal Artificial . . . . .	13
3.1.2.2. CNN . . . . .	17
3.2. Construcción, diseño y optimización de redes . . . . .	22
3.2.1. Entrenamiento . . . . .	22
3.2.1.1. Técnicas de optimización y regularización . . . . .	23
3.3. Computación Evolutiva . . . . .	27
3.3.1. Características y funcionamiento . . . . .	27
3.3.1.1. Codificación . . . . .	29
3.3.1.2. Operadores genéticos . . . . .	30
3.3.1.3. Reemplazo . . . . .	33
3.3.1.4. Evaluación . . . . .	33
3.3.2. Algoritmos evolutivos Multiobjetivo . . . . .	34
3.3.2.1. NSGA-II . . . . .	35
<b>4. Diseño y Metodología</b>	<b>37</b>
4.1. Herramientas . . . . .	37
4.1.1. Entorno de ejecución . . . . .	37

4.1.1.1.	CESGA	37
4.1.1.2.	Hardware	39
4.1.1.3.	Software	40
4.1.2.	Conjunto de datos	41
4.2.	Preprocesamiento de los datos	42
4.3.	Algoritmo genético	45
4.3.1.	Codificación de un individuo	45
4.3.2.	Evaluación. Cálculo de la <i>fitness</i>	47
4.4.	Estrategias de desarrollo	49
4.4.1.	Primera ejecución	49
4.4.2.	Segunda ejecución: entrenamiento distribuido	52
4.4.3.	Tercera ejecución: exploración vs. explotación	53
4.4.4.	Cuarta ejecución: ajuste de la <i>fitness</i>	56
4.4.4.1.	Precisión vs. Tamaño	56
4.4.4.2.	Capacidad de clasificación de la red	57
4.4.4.3.	Complejidad de una red	57
<b>5.</b>	<b>Resultados y Discusiones</b>	<b>59</b>
<b>6.</b>	<b>Conclusiones y Trabajos futuros</b>	<b>71</b>
<b>References</b>		<b>87</b>

# Índice de figuras

2.1. La plataforma robótica Flourish BoniRob con la herramienta de control propuesta por [1] . . . . .	4
3.1. Inteligencia artificial, aprendizaje automático y aprendizaje profundo. . . . .	11
3.2. IA simbólica vs. ML: el nacimiento de una nueva era [2] . . . . .	12
3.3. Cronología de los artículos principales y originales de los clasificadores de ML no probabilísticos más utilizados para la clasificación de imágenes. Imagen original de [3] . . . . .	12
3.4. Comportamiento del Perceptrón [4] . . . . .	13
3.5. Representación gráfica de las topologías de redes neuronales más populares (2019). Imagen de [5] . . . . .	14
3.6. Ejemplo de la estructura de una ANN. Imagen de [6] . . . . .	15
3.7. Funciones de activación más populares . . . . .	16
3.8. Ejemplo genérico de la estructura de una Red Neuronal Convolucional (CNN) . . . . .	17
3.9. Ejemplo de aplicar un kernel sobre una imagen RGB . . . . .	18
3.10. Ejemplo de aplicar un kernel o filtro $3 \times 3$ sobre una capa de entrada $5 \times 5$ con <i>stride</i> 1 . . . . .	18
3.11. Ejemplo de aplicar un kernel o filtro $3 \times 3$ sobre una capa de entrada $5 \times 5$ con <i>stride</i> 2 . . . . .	19
3.12. Ejemplo de <i>zero-padding</i> sobre una matriz $5 \times 5$ , obteniendo finalmente una matriz $7 \times 7$ . . . . .	19
3.13. Ejemplo de tres tipos de <i>pooling</i> ; <i>max</i> , <i>min</i> y <i>average pooling</i> . . . . .	19
3.14. Efecto sobre la pérdida al aplicar la técnica de <i>Batch Normalization</i> . . . . .	20
3.15. Ejemplo del funcionamiento de las capas totalmente conectadas y las capas de promedio global [7] . . . . .	20
3.16. Arquitectura de la VGG16. Imagen de [8] . . . . .	21
3.17. Arquitectura de GoogleNet. Imagen de [?] . . . . .	21
3.18. Arquitectura de la ResNet. Imagen de [9] . . . . .	22
3.19. Entrenamiento de una NN básica: Pesos, Función de pérdida, Optimizador y Retropropagación	23
3.20. Ratio de aprendizaje o Learning rate. Imagen de [10] . . . . .	23
3.21. Efecto en el entrenamiento del valor del ratio de aprendizaje definido . . . . .	24
3.22. Actualización del gradiente con el optimizador Momentum (izq.) y con Nesterov (dch.) [11] .	24
3.23. Ejemplo gráfico de las diferencias de actuación de algunos de los optimizadores más populares	25
3.24. Ejemplo gráfico de los posibles ajustes a realizar, incluido el sobreajuste u <i>overfitting</i> . . . . .	25
3.25. Ejemplo de <i>overfitting</i> y <i>underfitting</i> durante el proceso de entrenamiento . . . . .	26
3.26. Ciclo evolutivo genérico de los Algoritmos Evolutivos . . . . .	28
3.27. Ejemplos de algoritmos evolutivos y su codificación . . . . .	29
3.28. Ejemplo de codificación completa . . . . .	30
3.29. Ciclo evolutivo genérico de los Algoritmos Evolutivos . . . . .	30
3.30. Ejemplo de cruce en un punto . . . . .	32
3.31. Ejemplo de cruce en $N$ puntos, siendo $N = 2$ . . . . .	32
3.32. Ejemplo de cruce uniforme . . . . .	32
3.33. Ejemplo de aplicación del operador de mutación . . . . .	33

3.34. Ejemplo del flujo evolutivo de un problema de diseño de redes neuronales . . . . .	34
3.35. Ejemplo de diferentes frentes de Pareto para el mismo espacio de soluciones . . . . .	35
3.36. Esquema del mecanismo de promoción/preservación de individuos del NSGA-II . . . . .	36
4.1. Distribución del supercomputador FinisTerra III [12]. Imagen de [13] . . . . .	38
4.2. Captura de pantalla al ejecutar el comando para revisar los módulos disponibles en FinisTerra III. . . . .	39
4.3. Distribución geográfica de las imágenes de malas hierbas en el norte de Australia (Datos: Google, SIO, NOAA, U.S. Navy, NGA, GEBCO; Imagen Landsat/Copernicus 2018; Image DigitalGlobe 2018; Imagen CNES/Airbus 2018). Imagen de [14] . . . . .	41
4.4. Gráfico con la distribución de las clases en el dataset. . . . .	42
4.5. Gráfico con la distribución de las clases para cada subconjunto de datos tras la división del dataset original. A la izquierda el reparto resultante al seguir una división aleatoria estratificada, mientras que a la derecha se realiza una división meramente aleatoria simple. . . . .	43
4.6. Diferencia entre una capa residual básica y otra con una operación convolucional. Imagen de [15] . . . . .	47
4.7. <i>Fitness</i> del problema multiobjetivo . . . . .	48
4.8. Gráfico con los resultados al realizar 10 entrenamientos para evaluar el rendimiento del hardware disponible . . . . .	51
4.9. Ejemplo gráfico del funcionamiento del entrenamiento distribuido en una máquina con 2 GPUs .	52
4.10. Consecuencia de variar un único bit en la cadena o cromosoma de un individuo . . . . .	55
4.11. Ejemplo gráfico de que el orden de la <i>fitness</i> genera los mismos frentes de Pareto, aunque el orden de los individuos sí varía . . . . .	57
4.12. Ejemplo del entrenamiento de una red obtenida durante el proceso evolutivo . . . . .	58
4.13. Correlación de Spearman entre las variables de una arquitectura CNN. Imagen de [15] . . . . .	58
5.1. Ejemplo de la evolución del flujo evolutivo según el tamaño (izq.) y la precisión de las redes (dch.); en función del número de capas máxima a codificar . . . . .	60
5.2. Comparación de las topologías de los experimentos ECNN-02 y ECNN-03 . . . . .	62
5.3. Gráfico del entrenamiento y las métricas principales de la ECNN-1.8 . . . . .	63
5.4. Matriz de correlación de la red ECNN-1.8 . . . . .	64
5.5. Gráfico del entrenamiento y las métricas principales de la ECNN-0.1 . . . . .	65
5.6. Matriz de correlación de la red ECNN-0.1 . . . . .	66
5.7. Gráfico del entrenamiento y las métricas principales de la ECNN-1.1 . . . . .	67
5.8. Matriz de correlación de la red ECNN-1.1 . . . . .	68
5.9. Frente de Pareto de los datos: relación entre el nº de capas y el nº de parámetros en las redes . .	69
5.10. Frente de Pareto con las dos redes solución . . . . .	70

# Índice de tablas

4.1.	Especificaciones del software utilizado en el proyecto . . . . .	39
4.2.	Especificaciones del hardware utilizado en el proyecto . . . . .	39
4.3.	Distribución de clases dentro del conjunto de datos . . . . .	42
4.4.	Codificación de una capa de la CNN con el algoritmo genético . . . . .	45
4.5.	Operadores y valores de probabilidad utilizados en los experimentos finales . . . . .	49
4.6.	Resumen asociado a los experimentos y resultados representados en la Fig. 4.8 . . . . .	51
5.1.	Algunas de las redes generadas con el algoritmo evolutivo y sus métricas y características más representativas . . . . .	61
5.2.	Informe de clasificación de la red ECNN-1.8 . . . . .	63
5.3.	Informe de clasificación de la red ECNN-0.1 . . . . .	65
5.4.	Informe de clasificación de la red ECNN-1.1 . . . . .	67
5.5.	Complejidad de las redes . . . . .	70



# Capítulo 1

## Introducción

### 1.1. Motivación

La mejora de las tecnologías, así como su notorio aumento de la capacidad computacional, han derivado en un crecimiento desmesurado de las redes neuronales, lo que implica no solo un aumento del tamaño de las mismas, sino también de su profundidad y número o cantidad de parámetros y operaciones a realizar. Este aumento de los requerimientos de hardware y capacidad de cómputo disponible para ello, han hecho que la complejidad de las soluciones también aumente, produciendo un encarecimiento del proceso.

Muchas veces, se generan redes increíblemente complejas para resolver problemas, en este caso de clasificación, relativamente sencillos. Para evitar el sobre-entrenamiento o sobre-ajuste en los entrenamientos, se aplican técnicas de regularización para así esquivar el problema de falta de generalización en los modelos. Finalmente, se puede afirmar que actualmente en el mundo del aprendizaje profundo es común hacer lo que se conoce como "matar moscas a cañonazos", es decir, que se emplean de forma absurda y excesiva recursos costosos para realizar tareas o resolver problemas de baja complejidad; finalmente, se realiza un esfuerzo innecesario en la resolución de problemas capaces de solucionarse con modelos sumamente más simples.

Por eso mismo, en este proyecto se propone un método novedoso y único, basado en los procesos evolutivos para el diseño de redes convolucionales (CNN) más pequeñas, más simples, pero con un buen rendimiento, es decir, que mantengan su capacidad de clasificación haciendo un uso más eficiente de los recursos. Para ello se propone un método de diseño que se adapte y genere arquitecturas o estructuras acordes a la complejidad del problema a resolver.

El desarrollo evolutivo, sigue una estructura y comportamiento bioinspirada, basado en el algoritmo multiobjetivo y elitista NSGA-II. Se propone un diseño personalizado del algoritmo para este problema concreto, diseñando un tipo de codificación personalizada para el diseño de CNNs, y una función de evaluación que depende directamente del problema a resolver. Este valor de *fitness* se encarga de encontrar una solución acorde a los requisitos o requerimientos definidos, haciendo que el algoritmo converja en el óptimo global, y se obtenga a solución deseada.

Finalmente, este procedimiento se ejecuta para obtener modelos ya entrenados durante el proceso evolutivo que debido a su comportamiento eficiente, sean capaces de ser integrados en sistemas de hardware con una baja capacidad de cómputo, para ser aplicados en mecanismos autónomos o en tiempo real.

## **I.2. Objetivos**

Este trabajo tiene como objetivo último el desarrollo de un algoritmo con un enfoque evolutivo para diseñar redes neuronales convolucionales para la clasificación de imágenes. Se busca que estas redes sean pequeñas, simples y eficientes, con una alta capacidad de clasificación. A su vez, la realización de este proyecto conlleva otros fines más específicos:

- Estudio de las arquitecturas y modelos entrenados, así como de las arquitecturas básicas ya existentes. Puesta en práctica de los conocimientos adquiridos y análisis de los entrenamientos y resultados obtenidos.
- Estudio y familiarización con el desarrollo de técnicas de optimización multiobjetivos basados en estrategias evolutivas.
- Diseño, desarrollo y optimización del algoritmo de generación de arquitecturas convolucionales basado en el algoritmo genético NSGA-II.
- Integración y ejecución del algoritmo de forma remota en infraestructuras de computación de alto rendimiento, así como la familiarización y adaptación al nuevo entorno de trabajo.
- Análisis y comparación de las redes generadas por el algoritmo, así como de los resultados obtenidos. Verificación de la hipótesis establecida en función del problema a resolver.

## **I.3. Estructura**

La estructura de este documento se divide en seis capítulos diferenciados, incluyendo esta **introducción**, con los que se intenta dar una visión global del trabajo realizado.

1. Este documento comienza con el **estado del arte** donde se hace una investigación y análisis de la documentación, metodología y proyectos más actuales y vanguardistas que se encuentran relacionados de forma directa con el proyecto realizado.
2. En el **marco teórico** se presentan las bases teóricas necesarias para la comprensión del trabajo. Este apartado se desarrolla con el fin de que sirva como lectura complementaria o herramienta de apoyo para el lector, para el correcto entendimiento del resto de la memoria.
3. En el apartado de **diseño y metodología**, se describe la estrategia de trabajo seguida, así como todos los requisitos prácticos necesarios para su ejecución y su reproducibilidad.
4. En cuarto quinto lugar se presentan y se analizan en profundidad los **resultados** obtenidos.
5. Por último, se presentan las **conclusiones**, donde se exponen las lecciones aprendidas, así como las ideas generales y más importantes deducidas tras el desarrollo del proyecto. También se incluyen las **líneas futuras** donde se exponen futuros cambios, modificaciones o estrategias a seguir, para ampliar y mejorar los resultados obtenidos.

## Capítulo 2

# Estado del arte

### 2.1. Agricultura de Precisión

La agricultura se define como la actividad que ocupa el cultivo y recogida de las cosechas. Este fue uno de los mayores descubrimientos del hombre, permitiendo un acceso sencillo a los alimentos y provocando una transformación de grupos nómadas a sociedades sedentarias como en la actualidad; la agricultura forma parte de las principales actividades que conforman el sector primario [16].

Con el paso del tiempo, el aumento de los conocimientos y mejora de las tecnologías, este sector ha sufrido numerosas revoluciones para adaptarse a las necesidades de la población [17, 18]. Ciertas mejoras o transformaciones que han surgido en esta industria, también han supuesto un deterioro importante de la calidad del medio ambiente. Tal y como apuntan informes de la FAQ (*Food and Agriculture Organization of the United Nations*), en los próximos años habrá cambios en la agricultura y la alimentación tal cual la conocemos hoy en día; lo que deriva en la necesidad de una agricultura más sostenible, que respete y se adapte al medio ambiente [19, 20, 21]. A lo que se suma el continuo crecimiento de la población mundial, que puede llevar a un problema de falta de recursos alimentarios para la totalidad de la población.

Dentro de este sector, se encuentra la agricultura de precisión, un tipo de gestión heterogénea de los cultivos cuyo objetivo es mejorar la productividad agrícola con un bajo coste, es decir, aumentar la cantidad y calidad del producto con un menor impacto. O como define la sociedad internacional de agricultura de precisión (ISPA): *"La agricultura de precisión es una estrategia de gestión que recoge, procesa y analiza datos temporales, espaciales e individuales, y los combina con otras informaciones como apoyo al manejo y toma de decisiones de acuerdo con la variabilidad estimada, para mejorar la eficiencia en el uso de recursos, la productividad, la calidad, la rentabilidad y sostenibilidad de la producción agrícola"* [22].

#### 2.1.1. Métodos de control de malas hierbas

La principal motivación del uso de agricultura de precisión proviene de la diversidad dentro de los cultivos; esta se debe a la variabilidad del entorno, las condiciones climáticas, las metodologías de producción, etc. Existen diferentes aplicaciones comprendidas en estas actividades, como la estimación de la densidad de sembrado, la aplicación autónoma y adecuada de herbicidas, analizar y calcular la cantidad adecuada de fertilizantes, analizar el estado de los cultivos, etc [23, 24, 25, 26, 27, 28].

Una de las técnicas más importantes dentro de la agricultura de precisión, es la aplicación selectiva de agroquímicos. Estos métodos buscan aplicar y ajustar automáticamente la dosis de tratamiento necesario para cada cultivo, utilizándose de forma eficiente únicamente en las zonas realmente necesarias [29, 30]. Este problema constituye una de las cuestiones más importantes y populares de la actualidad.

Las malas hierbas, compiten con las plantas del cultivo por los recursos de la tierra, reduciendo el rendimiento del terreno y destinando parte de estos recursos a plantas perjudiciales para los cultivos. El orden correcto para aplicar eficientemente estos tratamientos consiste en: 1) localizar; 2) calcular la densidad; 3) clasificar e identificar la especie de planta; y 4) aplicar el tratamiento más adecuado.

### 2.1.1.1. Técnicas de tratamiento

Existen diferentes técnicas para el control y tratamiento de malas hierbas. Estas se dividen en cuatro categorías diferenciadas [31, 32]:

- **Técnicas mecánicas:** se aplican directamente sobre las malas hierbas para eliminarlas. Comprende técnicas como el entierro, el corte o la quema de estas plantas [33, 34].
- **Técnicas culturales:** no hacen uso de químicos, sino que se aplican en base al conocimiento del cultivo y la propia tierra. Algunas de estas son el labrado, el uso de falsas siembras o la modificación de la fecha de siembra [35, 36].
- **Técnicas biológicas:** compuesta por técnicas que emplean enemigos naturales para su eliminación y control; como el pastoreo, el uso de micoherbicidas o la aleopatía [37, 38, 39].
- **Técnicas químicas:** incluyen el uso y aplicación de herbicidas para la eliminación de las especies de malas hierbas [40, 41, 42]. Estas técnicas han ganado terreno en el mundo del tratamiento y control de plagas, por su alta eficacia y facilidad de uso y aplicación [43]. [44, 1]

Actualmente, ya se desarrollan vehículos con diferentes herramientas incorporadas, capaces de aplicar herbicidas o hasta técnicas mecánicas con una alta y efectiva capacidad selectiva para poder eliminar estas hierbas sin afectar al cultivo [44]. Un ejemplo de esto es el autómata Como el que presentan en [1] (Fig. 2.1).

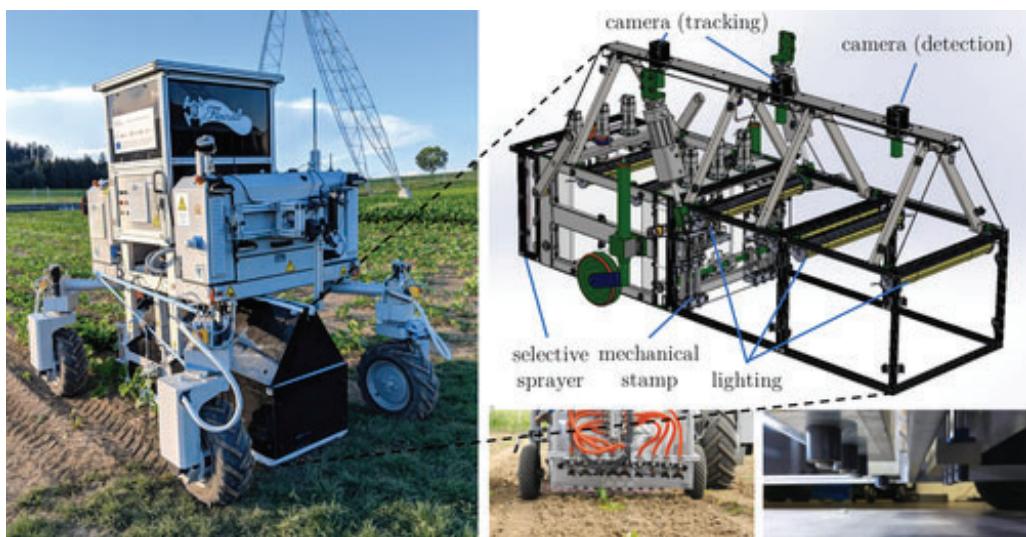


Figura 2.1: La plataforma robótica Flourish BoniRob con la herramienta de control propuesta por [1]

### 2.1.1.2. Técnicas de localización

La localización, identificación y clasificación de malas hierbas dentro del cultivo forma parte de uno de los retos más importantes y populares actualmente dentro de la agricultura de precisión [45, 46].

Uno de las principales dificultades del problema de identificación y clasificación de estas plantas es la alta variabilidad de especies y la complejidad de reconocimiento que esto conlleva, una ardua tarea hasta para ojos

expertos. Originariamente, este proceso se hacía *in situ* en los cultivos, donde se aplicaban los tratamientos tras un proceso de localización manual; debido a su alto coste, actualmente se aplican técnicas autónomas que facilitan y agilizan estos procesos de localización. Para ello, uno de los puntos cruciales para poder entrenar y evaluar estas técnicas es la disponibilidad de conjuntos de muestras de calidad, etiquetadas manualmente por uno o varios expertos de forma coherente y correcta [14, 46].

Algunas de las técnicas más desarrolladas a día de hoy consiste en el uso de cámaras hiper-espectrales [47] que se caracterizan por ser capaces de recopilar información de todo o gran parte del espectro electromagnético, incluidas frecuencias no visibles para el ojo humano [48, 49]. También es muy popular el uso de cámaras RGB capaces de detectar únicamente el espectro visible; también son significativamente más económicas que las anteriormente mencionadas, lo que promueve y populariza su uso. Estas cámaras son muy utilizadas y aplicadas junto con técnicas de visión por computador y aprendizaje profundo para diferenciar con una precisión alta los diferentes tipos de vegetación existentes en las imágenes [50, 51]. Un problema que tienen este tipo de cámaras es la gran influencia que ejerce la iluminación de la escena sobre la información de la imagen, haciendo que esta pueda llegar a perderse.

### 2.2. Métodos de Detección: clasificación de imágenes

La clasificación o reconocimiento de imágenes es una de las tareas principales en visión por computadora [52, 53, 54]; se usa en muchas áreas como la seguridad, ciencia, ingeniería, ciencia médica, agricultura, medio ambiente, marketing, y muchas otras áreas más. Existen numerosos ejemplos de clasificación de imágenes, como el etiquetado automático de imágenes de cáncer de piel [55]; o la clasificación de diferentes alimentos, como se hace en [56, 57, 58, 59]; también, se pueden realizar clasificaciones de imágenes capturadas con diferentes sensores, como las imágenes satelitales [60, 61, 62]; o como el tema de este proyecto, los problemas de identificación y reconocimiento agrícola [63, 64, 65, 66]; etc.

Hoy en día, las nuevas tecnologías están constante y masivamente generando datos e imágenes que es necesario clasificar para hacer que su manejo sea más rápido, fácil y accesible. Uno de los problemas más comunes de estas grandes cantidades de muestras o imágenes, es poder diferenciar clases, objetos o diferentes elementos que pertenezcan a una misma categoría, lo que se conoce como un problema multi-clase. Esto se refiere a la tarea de asociar una clasificación de etiqueta única, o más etiquetas (clasificación de etiquetas múltiples), a una imagen determinada [67, 68]. La principal diferencia entre la clasificación multi-clase y multi-etiqueta, es que el primer problema solo trabaja con una única etiqueta; mientras que los problemas multi-etiqueta, asocia cada una de las instancias a un conjunto de etiquetas determinado [69, 70]. En las tareas de clasificación, suele aparecer un problema comúnmente conocido como variación intra-clase que se refiere a la variabilidad de imágenes pertenecientes a una misma clase, como por ejemplo, la gran variedad de rostros que pueden tener las personas, cuya etiqueta final sería "humano"; este es uno de los principales desafíos en la clasificación de imágenes [71, 72, 73, 74].

En este proyecto, los principales problemas y tareas a analizar pertenecen a la categoría multi-clase de etiqueta única; siendo el propósito final de las redes discernir entre plantas "buenas" y "malas" a su vez, identificar la especie particular de maleza, como realiza Olsen et al. en [14].

#### 2.2.1. Aprendizaje automático

Dentro de esta categoría existen diferentes métodos y técnicas como los algoritmos de aprendizaje automático [75, 76, 3]. El ML es un área amplia, y algunos de sus algoritmos más populares y utilizados en problemas de clasificación de imágenes son:

- El **K-vecinos más cercano (k-NN)**, uno de los clasificadores más utilizados debido a su simplicidad y efectividad; aunque muy susceptible ante el ruido en los datos [3, 76]. Un ejemplo de aplicación de este método es [77], donde se aplica a un problema de clasificación de imágenes médicas, y se compara con

otras técnicas de ML como el SVM, DT o Naive Bayes. Los resultados obtenidos por Singh et al. confirma que el k-NN es el algoritmo que mejor se adapta a este problema, dando un valor de 5 al parámetro  $k$  y utilizando el peso de la distancia inversa al cuadrado.

- Los **árboles de clasificación o decisión** [78, 79, 80], algoritmo de aprendizaje supervisado que se caracteriza por hacer una división recursiva del espacio de instancias, dando lugar a estructuras fáciles de comprender e interpretar, y aptos para trabajar con conjuntos de datos incompletos; por otro lado, presenta problemas para controlar el tamaño de los árboles resultantes. Existen diferentes modificaciones de este algoritmo, como el árbol de decisión híbrido (HDT) presentado en [81], entre otros [82, 83]. También pueden ser susceptibles al *overfitting*, para lo que se crea el RFA o **random forest algorithm** [84, 85, 86, 87].
- El SVM o **support vector machine** [88], es uno de los métodos de predicción más robustos, cuyo origen se debe al aumento excesivo de la cantidad de datos disponibles. Es un clasificador lineal binario no probabilístico, capaz de clasificar datos lineales y no lineales; construye un hiperplano como límite de decisión que distingue los diferentes grupos presentes en los datos. Tiene varias aplicaciones [89, 3, 90, 91, 92, 93]. El uso de este algoritmo supone un algo coste computacional y un largo tiempo de procesamiento; no responde correctamente con conjuntos de datos ruidosos. Originalmente, SVM solo funciona con dos clases, limitación superada con la modificación Multi-class SVMs [94, 95, 96].

En los últimos años, el **aprendizaje profundo** o *deep learning* ha comenzado a tomar mucha importancia dentro del campo de la informática, entre muchos otros [97, 98, 99]. Una de las muchas aplicaciones de estas técnicas es la clasificación de imágenes [100, 101, 102]. Por ejemplo, Dhillon et al. hacer una comparación entre métodos de DL (DNN) y algoritmos de ML (RF) para la clasificación de imágenes hiperespectrales [103]. Dentro del DL existen diferentes modelos como las redes neuronales recurrentes (Recurrent Neural Networks, RNN) [104], las redes generativas antagónicas (Generative adversarial network, GAN) [105], el *restricted Boltzmann machine* (RBM) [106], etc. Dentro del ámbito del reconocimiento de imágenes, las técnicas de DL más utilizadas son las redes convolucionales (CNN).

### 2.2.1.1. Aprendizaje automático para la detección de malas hierbas

Existen diferentes artículos y revisiones sobre la aplicación y comparación de diferentes algoritmos de ML y modelos de DL aplicados a la clasificación de imágenes, más específicamente para la detección de malas hierbas [107, 108, 109, 110, 111, 112].

Todas las técnicas mencionadas anteriormente se utilizan para la detección de malezas, dando lugar a numerosos estudios de comparación de rendimiento y eficiencia entre métodos [113, 114, 115]; también existen otros algoritmos como Naïve Bayes [116, 117], clasificadores bayesianos [118], o AdaBoost [119, 120], que también se utilizan pero en menor medida.

La presencia y aplicación del **algoritmo kNN** en la detección de malezas es moderada, como se explicó anteriormente, esta técnica es muy fácil de implementar aunque sus resultados en clasificación no sean los mejores; sin embargo, se aplica en diferentes proyectos [121], p. ej., en un artículo relativamente reciente [122], se aplica kNN para diferenciar y clasificar malezas y cultivos de remolacha; analizando los resultados en función de los valores de precisión y el tiempo de ejecución. Otro ejemplo es [123], donde para reducir el tiempo de procesamiento y aumentar la precisión de detección en los sistemas aéreos no tripulados (UAS), desarrollan un enfoque de detección automático e inteligente basado en un novedoso algoritmo de vecinos más cercanos K de dos niveles (TLKNN). Guo et al. use utilizar una CNN para la extracción de características, para finalmente entrenar dos modelos kNN a partir de los cuales se obtienen los resultados de clasificación. La aplicación más reciente de kNN para la detección de malezas es presentada por Sinlae et al. [124] que tiene como objetivo construir un sistema de clasificación de malezas de hoja grande usando una combinación de los algoritmos kNN y PCA [125]; utilizando en esta ocasión el PCA para la extracción de características.

También, los **RFA**s son ampliamente utilizados en la detección de malezas. Por ejemplo, [126] propone un algoritmo para extraer características de textura y color de las imágenes, para luego usar un RFA para entrenar un modelo; este modelo se evalúa calculando métricas de regresión, precisión, recuperación y puntajes F1 y logra una precisión de clasificación correcta del 91 % para maleza, 100 % para suelo, 90 % para pasto y 99 % para cultivo de soja. Otro ejemplo de aplicación de RFA se presenta en [127], el algoritmo OBIA, una combinación de modelos digitales de superficie (DSM), ortomosaicos y RFA, consiguiendo un procedimiento automático, eficiente en el tiempo y muy preciso.

Sin embargo, los métodos más utilizados para la detección de malezas son el SVM y las CNNs [128, 129, 130]. El método **SVM** puede resolver problemas no lineales y muestra un buen rendimiento en reconocimiento de patrones con problemas con pocos datos o muestras [131]. Un ejemplo de esto es [132, 133, 134], donde se propone un método combinado con SVM para identificar y detectar la posición de plantas de maíz y maleza, y así lograr una fertilización más precisa. Otro ejemplo es [135], propusieron un sistema automatizado de detección de malas hierbas, compuesto por un primer bloque con métodos de extracción de características, junto con algoritmos de ML como la regresión logística y el SVM. En [136], Sneha Soni propone un método combinado; en primer lugar, se hace un agrupamiento con K-means para clasificar objetos en la imagen y detectar el color verde, luego se usa SVM para identificar y seleccionar la maleza. Por último, se presenta un nuevo enfoque híbrido de optimización de malezas invasivas [137]. Este método utiliza tres algoritmos de clasificación, kNN, SVM y RF, que se entrena mediante validación cruzada k-fold; esta combinación aumenta la precisión media del modelo, consiguiendo más del 99,90 % de precisión, y reduce considerablemente el tiempo de entrenamiento.

Las **CNNs** tienen una gran capacidad de aprendizaje y pueden clasificar datos desconocidos [138, 139]. En [140], Zhuang et al. hacen una evaluación de diferentes CNNs profundos para la detección de malezas en cultivos de trigo; comparan diferentes arquitecturas como CenterNet, Faster R-CNN, TridentNet, VFNet, YOLOv3, AlexNet, DenseNet, ResNet y VGGNet, ya entrenadas con conjuntos de datos pequeños y grandes. Todas las clasificaciones obtienen puntuaciones de precisión muy altas, superiores al 99 %; finalmente, concluyen que CenterNet, Faster R-CNN, TridentNet, VFNet y YOLOv3 son insuficientes, mientras que AlexNet, DenseNet, ResNet y VGGNet entrenados con un gran conjunto de datos de entrenamiento son muy efectivas. En la misma línea de estudio, [138] compara diferentes métodos para la clasificación de malezas y demuestra cómo las CNNs no son el único método eficiente, revisando diferentes técnicas de ML; finalmente demuestran que las CNNs, en concreto la cNET, consiguen los mejores resultados, teniendo un mejor desempeño durante el problema. En [141, 142] se presentan ejemplos simples del uso de CNNs para un sistema inteligente de detección de malezas. Otros aspectos importantes a considerar son algunas situaciones y condiciones como la influencia de la calidad de la imagen y la consistencia en el desempeño de la CNN para el mapeo de malezas [143]. Hu et al. simula las degradaciones sufridas en imágenes tras una reducción de su resolución, también aplica técnicas de sobreexposición, desenfoque y ruido; demuestra como los modelos de CNN entrenados con imágenes de baja calidad son más tolerantes contra la falta o ausencia de calidad en imágenes desconocidas o de evaluación. Además, demuestran cómo Faster R-CNN y Mask R-CNN son moderadamente tolerantes a niveles bajos de degradación de la calidad, a diferencia de Deeplab-v3 que es sorprendentemente robusta, pudiendo tolerar estas degradaciones en todos los niveles probados; también establecen un umbral de calidad para ayudar y guiar al usuario en la selección de cámaras para futuras aplicaciones de mapeo de malezas. En [?] se presenta una evaluación de cámaras y distancias en la imagen, para la detección de malas hierbas en cultivos de arándanos silvestres. Además, hay numerosos trabajos que utilizan y aplican la técnica del **transfer learning** [144, 145, 146, 147]. Por ejemplo, [148] busca un entrenamiento efectivo con la arquitectura CNN para conjuntos de datos pequeños y limitados. En este estudio, desarrollan una CNN transferible parcial para hacer frente a un nuevo conjunto de datos con diferente resolución espacial, el número de bandas y la variación en la relación señal-ruido, para hacer menos exigente el proceso de entrenamiento para nuevos conjuntos de datos.

### 2.2.2. Computación evolutiva

Los algoritmos genéticos son métodos bioinspirados altamente estudiados a lo largo del tiempo [149, 150, 151, 152, 153], y muy utilizados en problemas de optimización [154, 155, 156]. Estos métodos se han utilizado y aplicado en diferentes áreas como la ingeniería [157], medicina [158, 159], administración [160], incluida la clasificación de imágenes [161, 162, 163, 164]. También se utiliza muy frecuentemente para la optimización de hiperparámetros [165, 166, 167].

Uno de los usos más populares de los enfoques evolutivos en los problemas de visión por computador, es el reconocimiento de objetos o segmentación de la imagen; como es el ejemplo de [168] donde aplican proponen un nuevo método integrado basado en la programación genética para la clasificación de imágenes. Otro uso muy común de estos métodos es la segmentación de colores en la imagen, es decir, la identificación y localización de un color particular, como se propone en [169]; Senthil et al. presentan un algoritmo de segmentación para imágenes médicas, en concreto de cáncer de pulmón, para facilitar y agilizar su análisis. En [170] se propone un algoritmo genético mejorado para realizar una segmentación de la imagen en diferentes umbrales a partir de histogramas acumulativos. En esta misma línea, [171] propone un algoritmo genético híbrido, que junto con el algoritmo k-means y el filtro de Sobel, se aplica un proceso iterativo hasta que no se pueden encontrar dos regiones vecinas lo suficientemente similares. En [168] se basan en la idea de la aplicación de la programación genética para la extracción automática de características y la clasificación de imágenes, proponiendo una mejora sustancial con un enfoque de programación genética multicapa (MLGP) para la extracción y clasificación automática de características de alto nivel. Por último, en [172] proponen una nueva técnica para usar la transferencia de aprendizaje (*transfer learning*) en algoritmos con enfoques evolutivos para aprender problemas de clasificación de imágenes.

#### 2.2.2.1. CE para la detección de malas hierbas

La computación evolutiva también tiene cabida en el área de la agricultura de precisión [173, 174]; incluida la detección de malas hierbas. Tang et al. aplican un algoritmo genético para la segmentación supervisada de imágenes en color [175]; para la detección de malas hierbas utilizan un algoritmo genético codificado en binario que identifica regiones del espacio de color según la intensidad de saturación del tono (GAHSI). En el caso de [176], proponen un enfoque de segmentación automática basado en un algoritmo genético y una transformada de Radon para detectar líneas de cultivo de caña de azúcar a partir de imágenes RGB obtenidas a partir de un UAV; aunque esta propuesta se enfoca en la extracción de cultivos, pero esto sería extrapolable a la segmentación de malas hierbas.

## 2.3. Aprendizaje profundo evolutivo

El aprendizaje profundo evolutivo o *evolutionary deep learning*, es una metodología muy reciente en la que se hace uso de estrategias evolutivas, lo que incluye el proceso iterativo evolutivo bioinspirado que hace uso de poblaciones e individuos sobre los que se aplican diferentes operadores genéticos, para el diseño de redes neuronales [177, 178, 179].

Existen numerosos proyectos que se basan en la programación genética para este fin, como hacen en [180] donde se propone un nuevo método de DL evolutivo, basado en la programación genética con operadores de convolución para el aprendizaje o extracción de características en la clasificación de imágenes binarias multiclase. La programación genética se representa en forma de árbol, donde cada uno de los nodos representa un tipo de capa, y las aristas hacen referencia a las conexiones de capas de las redes. Esta estructura flexible permite que el algoritmo desarrolle programas simples para tareas fáciles y programas más complejos con múltiples operadores convolucionales para problemas más difíciles; en esta misma línea se presentan más trabajos como [181, 182]. En el libro [183], dedican un capítulo al desarrollo de un nuevo enfoque EDL basado en la GP para la clasificación de imágenes. Esta metodología también utiliza árboles con un nuevo conjunto de funciones y terminales que

representarán el tipo de capas y posibles conexiones entre estas; estas representaciones son fáciles de entender e interpretar. Este método de Bi et al. utiliza la precisión de clasificación promedio como la función objetivo; esta metodología se pone a prueba con 6 conjuntos de datos de imágenes faciales diferentes.

Por último, se deja a un lado la programación genética para poner el foco en los algoritmos genéticos, donde los individuos se codifican con cadenas de bits; este es el caso más interesante y similar al procedimiento seguido en este proyecto. El primer caso que se encuentra que sigue este procedimiento, es [184, 183], donde se propone un nuevo método que utiliza algoritmos genéticos para optimizar las arquitecturas de las redes, así como los valores de los pesos en el proceso de inicialización. Se enfocan en los problemas de clasificación de imágenes, para ello utilizan un algoritmo genético con una codificación de genes de longitud variable que representa los componentes básicos de la red, así como la profundidad y tamaño de la misma. La codificación propuesta por Sun et al. se caracteriza por posibilitar la existencia de genes con longitudes variables que pueden codificar hasta tres capas diferentes (capas convolucionales, capas *pooling* y capas densamente conectadas). También proponen un nuevo operador genético de longitud variable también, y como *fitness* utilizan un método nuevo propuesto por ellos, compuesto tres indicadores, el valor medio, la derivación estándar y el número de parámetros. En la misma línea, se propone el EvoDCNN [185], es un modelo evolutivo basado en bloques para desarrollar una red convolucional profunda evolutiva para la clasificación de imágenes. En este caso se utiliza una codificación lineal, en forma de cadena con una longitud fija pero con la posibilidad de generar redes con dimensiones diferentes. En este caso, únicamente utilizan la métrica del entrenamiento de precisión (*accuracy*) como valor de *fitness*. En referencia a la codificación utilizada por Hassanzadeh et al. tiene dos parámetros críticos, el número de bloques y el número de capas de convolución; proponen un tamaño de 5 capas por bloque, pero si este se modificaba, se incrementa la longitud de la codificación automáticamente, lo que hace que esta codificación se caracterice por su alta adaptabilidad y flexibilidad.



# Capítulo 3

## Marco Teórico

### 3.1. Inteligencia Artificial

En los últimos años, el aprendizaje profundo ha comenzado a tomar mucha importancia en diversos campos, como es el campo computacional. Primero, para comprender qué es el aprendizaje profundo y cómo funciona, debemos definir dos conceptos esenciales: el aprendizaje automático y la inteligencia artificial.

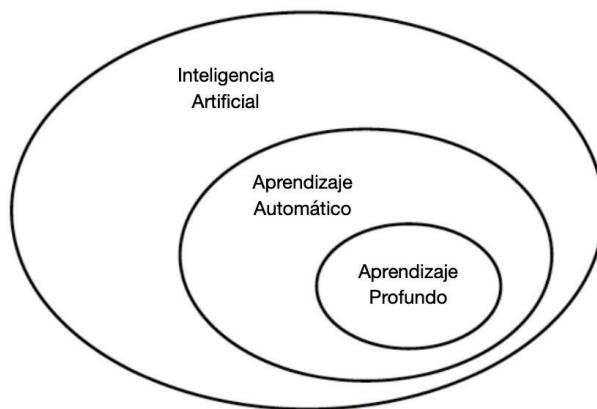


Figura 3.1: Inteligencia artificial, aprendizaje automático y aprendizaje profundo.

La inteligencia artificial nació en la década de los 50 cuando el principal tema de estudio en el campo de la informática estaba enfocado en el enigma de la existencia de pensamiento propio en máquinas. Para François Chollet la IA se puede definir como [2]: “el esfuerzo por automatizar tareas intelectuales normalmente realizadas por humanos”. La inteligencia artificial es un campo amplio y general que comprende, entre muchos otros más, al aprendizaje automático y el aprendizaje profundo, como se representa en 3.1.

El primer enfoque que surgió en este campo se conoce como IA simbólica, muy popular entre las décadas de 1950 y 1980; consistía en el diseño de reglas codificadas y respuestas que se utilizaban como resultados. La IA simbólica surgió como un método idóneo para la resolución de problemas lógicos como el ajedrez. Este fue el comienzo de una nueva y más compleja línea de investigación conocida como aprendizaje automático, disciplina científica comprendida dentro del campo de la IA.

### 3.1.1. Aprendizaje automático

El aprendizaje automático o ML podría definirse como un método de análisis de datos que automatiza los procedimientos de construcción de modelos analíticos, basado en la idea de que las máquinas piensan por sí mismas, por lo que la participación e intervención humana en estos procesos es mínima [2]. El interés por este campo fue creciendo y con él surgieron más inquietudes y preguntas; se buscaba algo más complejo que una mera herramienta que imitara al ser humano, es decir, un instrumento que no solo ayude al ser humano con lo que ya sabe hacer.

La principal diferencia entre la IA simbólica y el aprendizaje automático es que en el primero se usan las máquinas como una herramienta de apoyo y ayuda, mientras que el propósito del ML es que las máquinas aprendan de su propia experiencia, es decir, por sí mismas. Las reglas resultantes del proceso de aprendizaje del ML, se pueden generalizar y aplicar a nuevos datos.

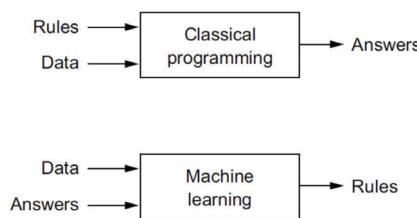


Figura 3.2: IA simbólica vs. ML: el nacimiento de una nueva era [2]

Dentro del ML, existen diferentes tipos de técnicas de aprendizaje dependiendo del método de adquisición de conocimiento [186, 187]:

- **Aprendizaje supervisado:** es la forma más común de aprendizaje automático. Se parte de pares de datos/solución. Funciona con datos etiquetados, intenta encontrar una función o modelo para clasificar y etiquetar correctamente los datos de entrada; el propósito es predecir la etiqueta de salida a partir de un histórico de datos. Algunos algoritmos que utilizan el aprendizaje supervisado son los árboles de decisión (*Classification trees*), Naïve Bayes, la regresión lineal y logística, K-vecino más cercano (*K-Nearest Neighbor*) o SVM (*Support Vector Machine*).
- **Aprendizaje no supervisado:** funciona con datos no etiquetados. Durante el entrenamiento, el algoritmo de forma autónoma adapta sus propias características para comprender e interpretar la información. Los algoritmos no supervisados catalogan los datos y trata de clasificarlos en grupos donde sus integrantes son lo más homogéneos posible entre sí, sin tener la capacidad de definir cómo es cada individualidad de cada uno de estos, y lo más heterogéneos posible con otros grupos. Algunos algoritmos que utilizan el aprendizaje no supervisado son métodos de agrupamiento (*Clustering methods*), métodos para la detección de anomalías, o de reducción de dimensionalidad, etc.

Dentro del aprendizaje automático existen numerosos algoritmos utilizados para diferentes tareas y áreas de estudio, recopilados en numerosos resúmenes [188, 3, 189, 190, 191]. Algunos de los algoritmos de ML más populares se muestran en la Fig. 3.3.

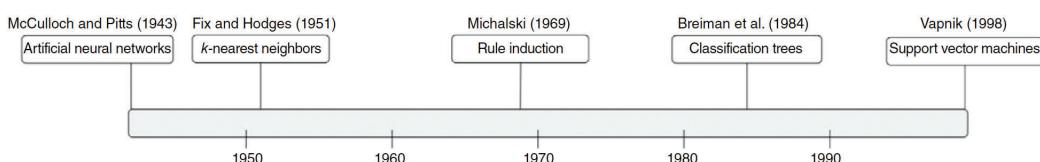


Figura 3.3: Cronología de los artículos principales y originales de los clasificadores de ML no probabilísticos más utilizados para la clasificación de imágenes. Imagen original de [3]

### 3.1.2. Aprendizaje Profundo

Partiendo de lo visto anteriormente (Sec. 3.1.1), el aprendizaje profundo se puede definir como un conjunto de algoritmos de ML, un método de aprendizaje artificial, sintético y bio-inspirado, inspirado en el sistema nervioso natural, que intenta imitar el comportamiento humano clasificando e identificando patrones para mejorar y aprender por sí mismo. El término DL surge en la década de 1980 pero hasta la década del 2010 no alcanzó su punto álgido, consecuencia directa de las limitaciones tecnológicas del momento. Hoy en día, en la era de internet y el *big data*, son el aliado perfecto ya que crean un ecosistema ideal que motiva y favorece el desarrollo del DL, convirtiéndolo así en una de las herramientas más utilizadas en la actualidad.

#### 3.1.2.1. Red Neuronal Artificial

Como se ha explicado, el aprendizaje profundo es un subgrupo dentro del aprendizaje automático, que intenta imitar y reproducir el comportamiento del cerebro humano; es un conjunto de técnicas y algoritmos cuyo fin último es obtener modelos abstractos capaces de identificar y extraer características de los datos mediante transformaciones matemáticas no lineales.

Una red neuronal artificial o ANN [192, 193] es una estructura o mecanismo bioinspirado, diseñado para simular la forma en que el cerebro humano analiza y procesa la información. Las ANNs son la base de la IA y resuelve problemas que resultarían imposibles o difíciles para los estándares humanos o estadísticos, ya que poseen capacidades de autoaprendizaje.

**Perceptrón** Al igual que en el cerebro humano, una ANN se basa en una colección de unidades o nodos conectados llamados neuronas artificiales. El perceptrón es la red más pequeña y simple, se conoce como una red neuronal de 1 sola capa y fue creada por el psicólogo estadounidense Frank Rosenblatt en 1957 [194]. Este sistema consta de una unidad o neurona artificial, que recibe como entrada un conjunto de elementos de procesamiento  $n_1$  y  $n_2$ , con un parámetro de peso  $w_n$  por cada entrada; finalmente se obtiene la salida  $n_f$  o  $y$  (Fig. 3.4).

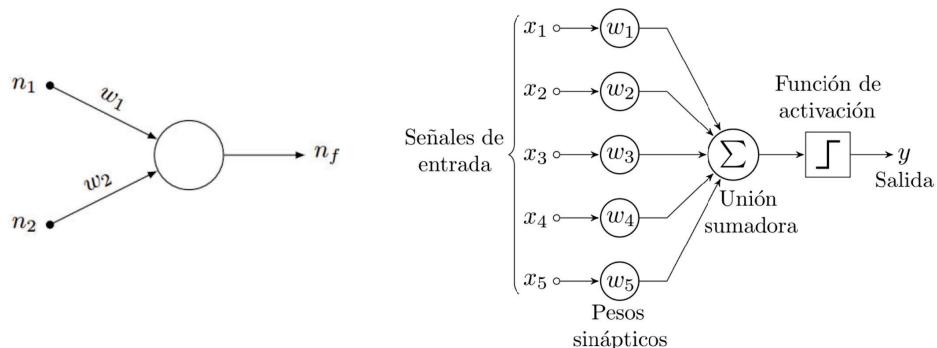


Figura 3.4: Comportamiento del Perceptrón [4]

El funcionamiento de este sistema es muy simple. Como entrada, la neurona recibe un escalar  $x_n$  que se multiplica con su respectivo peso  $w_n$ ; el resultado de esta operación constituye la entrada a una unión sumadora, donde también se debe tener en cuenta un peso fijo ligado a cada entrada, lo que se conoce como *bias* ( $b$ ). La última operación que se realiza en la neurona consiste en aplicar una función de transferencia o de activación ( $f$ ) sobre la salida obtenida del sumatorio, como se ve gráficamente en la Fig. 3.4, y matemáticamente en la Ec. 3.1.2.1.

$$\begin{aligned} net_n &= \sum (x_n * w_n + b_n) \\ y_n &= f(net_n) \end{aligned} \tag{3.1}$$

El perceptrón solo puede resolver funciones lineales separables, una limitación que se superó con los perceptrones multicapa [2], formados por la sucesión de varios perceptrones simples. Estas redes están compuestas por conexiones entre neuronas agrupadas en diferentes niveles que se conocen como capas; estas conexiones entre diferentes capas de neuronas permiten representar funciones no lineales más complejas.

**Multicapa** Según François Chollet [2], una ANN es: “*como una operación de destilación de información en varias etapas, donde la información pasa por sucesivos filtros y sale cada vez más depurada*”. Su propósito final es hacer que el proceso de entrada-objetivo se ejecute correctamente, con una secuencia profunda de transformaciones de datos simples que la red aprende a partir de ejemplos. En los perceptrones multicapa, el resultado de la función de activación se enviará a niveles más profundos.

En función del problema que se intente resolver, la interconexión se realizará de diferentes maneras dando lugar a diversas topologías en función de las necesidades de la red. En la Fig. 3.5 se representan algunas de las topologías más utilizadas.

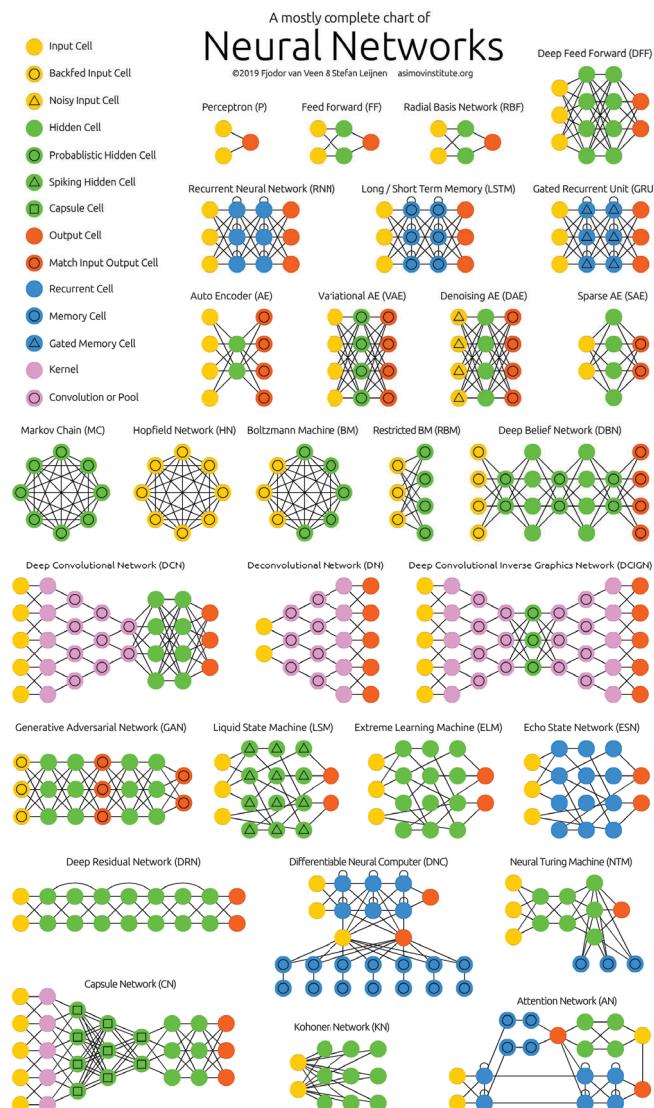


Figura 3.5: Representación gráfica de las topologías de redes neuronales más populares (2019). Imagen de [5]

Se entiende como topología de una red la organización y disposición de las neuronas, formando capas o agrupaciones de neuronas. Estas capas se caracterizan por su grado de profundidad, dado por el número de neuronas que la constituyen, siendo más profundas cuantas más tenga, y viceversa. Dentro de las capas, se pueden distinguir tres tipos (Fig. 3.6):

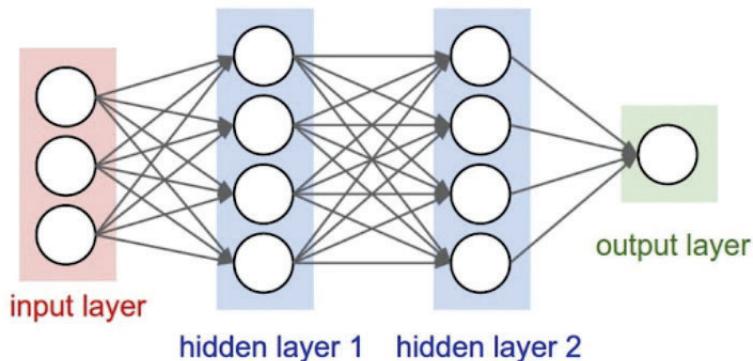


Figura 3.6: Ejemplo de la estructura de una ANN. Imagen de [6]

- **Capa de entrada (Input Layer):** su función es recibir los valores de entrada para alimentar la red con ellos. Los nodos en este paso son pasivos, lo que significa que no cambian los datos, no realizan ningún tipo de procesamiento. En esta capa se encuentran los datos de entrada y su profundidad depende de la cantidad de los mismos.
- **Capa oculta (Hidden Layer):** dentro de estas capas se encuentran las neuronas; estas aplican transformaciones a los valores de entrada que recibe la capa. Una capa oculta está conectada con la capa de salida o con otra capa oculta (conexiones ponderadas); no tienen interconexión directa con el entorno. Los valores que ingresan a un nodo oculto se multiplican por los pesos, para finalmente sumarse y producir un único valor numérico. El número de capas y su profundidad dependerán del problema que se quiera resolver, a mayor dificultad, mayor profundidad.
- **Capa de salida (Output Layer):** esta capa corresponde con el final de la red neuronal y puede estar conectada con capas ocultas o directamente con la capa de entrada. Tras combinar y modificar los valores de entrada, se devuelve uno o varios valores de salida que corresponden con la predicción de la variable respuesta; en esta capa habrá tantas neuronas como etiquetas o valores de salida.

Los parámetros fundamentales de la red son: número de capas, número de neuronas por capa, grado de conectividad y tipo de conexión entre neuronas.

**Función de activación** La función de transferencia o activación, comentada anteriormente, se encarga de devolver una salida a partir de un valor de entrada. Normalmente, el conjunto de valores de salida se encuentra dentro de un rango determinado como  $[0, 1]$  o  $[-1, 1]$ ; también es interesante la elección de la función, ya que tiene un gran impacto en la capacidad y rendimiento de la red neuronal, por eso mismo se busca que su derivada sea simple y fácil de calcular. La función de activación es la misma para todas las neuronas de una misma capa, esta puede diferir de la función aplicada a las neuronas de otras capas.

- **ReLU:** es una función simple, rápida y con un coste computacional bajo, de ahí su gran popularidad (Ec. 3.1.2.1). Existen variaciones de ReLU como la función Leaky ReLU/PReLU. Los valores de salida se encuentran en un rango de  $[0, +\infty]$ .

$$f(x) = \max(0, x) \quad (3.2)$$

- **Softplus:** es una aproximación suave de la función ReLU y puede ser usada para restringir la salida de una neurona para que siempre sea positiva. Es una expresión más compleja lo que implica un mayor coste computacional (Ec. 3.1.2.1). Los valores de salida se encuentran en un rango de  $[0, +\infty]$ .

$$f(x) = \log(e^x) \quad (3.3)$$

- **Tangente hiperbólica (tanh):** prácticamente idéntica a la función Sigmoide excepto porque está centrada (Fig. 3.7c). La salida de la función tendrá una media aproximadamente de 0 y se encuentra en un rango de entre  $[-1, 1]$  (Ec. 3.1.2.1). Usualmente, la convergencia será más rápida si el promedio de cada variable de entrada es cercano a cero, por lo que este modelo convergerá rápidamente. Función muy utilizada por la derivabilidad en todo su rango.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.4)$$

- **Sigmoide:** similar a la tangente hiperbólica; ambas funciones comparten la propiedad de ser derivable en todo su rango. Esta función puede saturar, es decir, si la entrada es muy grande o pequeña, el gradiente de la función sigmoide se acerca a 0, por lo que no hay un gradiente que fluya para actualizar los parámetros, haciendo que la red entrenada converja en un óptimo local. Los valores de salida se encuentran en un rango entre  $[0, 1]$  (Ec. 3.1.2.1).

$$f(x) = \frac{1}{1 + e^{-x}} \quad (3.5)$$

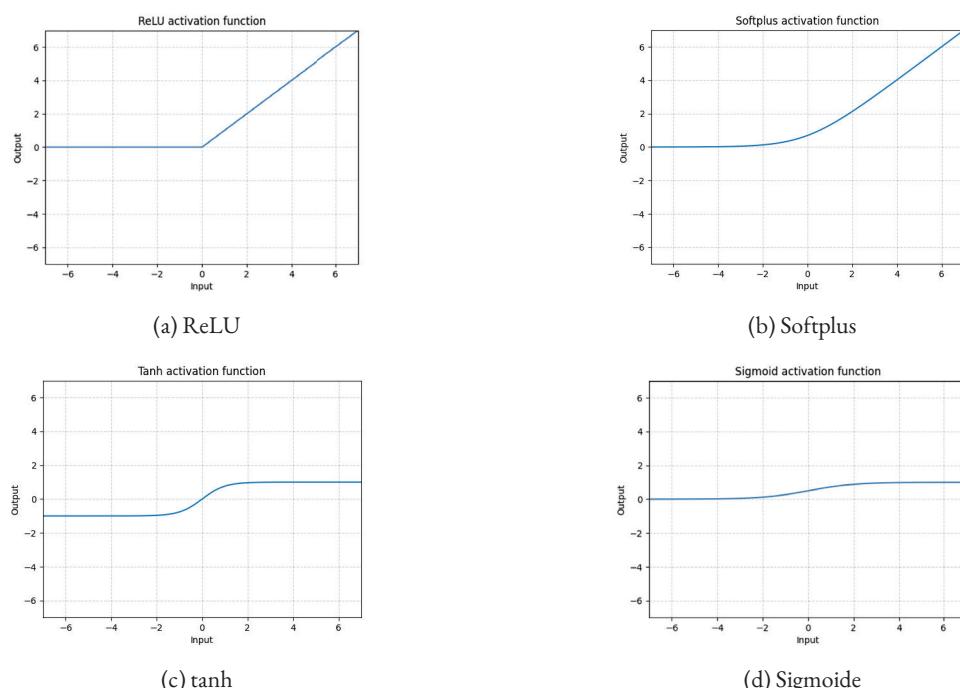


Figura 3.7: Funciones de activación más populares

### 3.1.2.2. CNN

Las Redes Neuronales Convolucionales o CNNs, son un tipo de perceptrón multicapa, muy útiles para tareas de visión por computador (*computer vision*). El diseño de estas redes se sigue asemejando al comportamiento de las neuronas de nuestro cerebro; en este caso, las CNNs se inspiran principalmente en el comportamiento realizado por el córtex visual para sacar información a partir de las imágenes, llegando a detectar hasta dependencias espaciales y temporales de una imagen.

El concepto de visión por computador o *computer vision* consiste principalmente en la obtención de información a partir de imágenes; existen numerosos retos en este área del DL, como la reconstrucción de imágenes, la segmentación, el reconocimiento o clasificación, o hasta el seguimiento de personas u objetos en secuencias de vídeo o imágenes digitales [195]. Estos retos no son triviales ya que las imágenes de personas u objetos presentan cierta dificultad, no solo para las máquinas sino hasta para el ojo humano (alta variabilidad intra-clase; problemas de iluminación, contraste o sombras; alta variabilidad geométrica, escalas, rotaciones, etc.; recorte u occlusión de objetos; deformaciones; etc.). Al trabajar con imágenes cada píxel se convierte en una entrada, considerando que existen diferentes tipos de imágenes (imágenes multicanal como RGB, CMYK, HSV, imágenes multiespectrales, etc.), se debe tener en cuenta cada uno de sus canales, multiplicando así las entradas o *inputs* de la red. Esta gran cantidad de datos de entrada y alta demanda de recursos computacionales son muy difíciles de manejar, incluso intratables, para una red simple, dando paso a las CNNs que se caracterizan principalmente por su alta capacidad de trabajo con matrices bidimensionales y extracción de información; su capacidad de manejo de grandes dimensiones de datos y diferentes topologías de píxeles; y el cierto grado de invariancia que presenta ante dificultades como la iluminación, cambios geométricos y otros.

**Estructura** Como se ha comentado, las CNNs fueron un gran avance en el campo de la visión por computador, demostrando grandes ventajas como:

- Alta capacidad de trabajo con grandes dimensiones de datos.
- Gran capacidad de trabajo con topologías de píxeles variadas (2D/3D).
- Invariancia ante dificultades lumínicas y geométricas.

Existen diferentes tipos de capas esenciales que componen la arquitectura de una red convolucional [196, 197]; cada una de estas capas se encarga de realizar ciertas operaciones específicas. Estas redes están formadas principalmente por dos partes o fases, la extracción de características (*feature extraction* o *feature learning*), encargada de la descripción de la imagen, y la clasificación (*classification*). En la primera parte, se aplican filtros convolucionales para la búsqueda de características, extrayendo algunas de las más importantes y básicas como bordes, patrones o formas; a continuación, se utilizan algunos métodos para reducir el tamaño y dimensiones de los datos, pero manteniendo sus características e información. Este método de reducción, generalmente se aplica tras las capas convolucionales, para mitigar o disminuir la carga computacional de las operaciones. Finalmente, se dispone la fase de clasificación, como se aprecia en la Fig. 3.8.

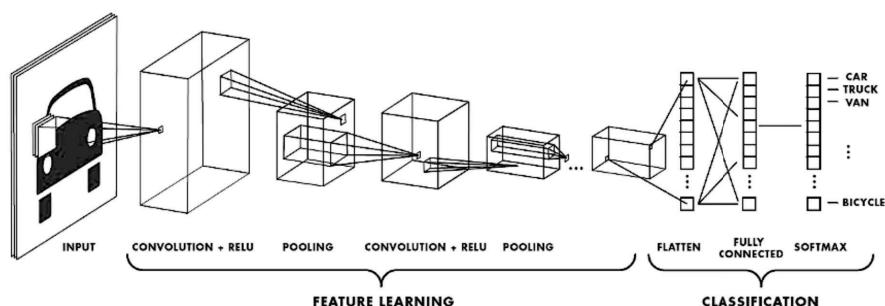


Figura 3.8: Ejemplo genérico de la estructura de una Red Neuronal Convolucional (CNN)

Como ya se ha comentado, las redes convoluciones se caracterizan por trabajar con matrices bidimensionales. Las imágenes están compuestas por píxeles; estas a su vez pueden ser multicanal es decir, una superposición de imágenes o planos. Para entender las capas convolucionales y su funcionamiento, hay que introducir un nuevo término, conocido como kernel (Fig. 3.9 y 3.10). Estas capas trabajan con matrices, recibe una matriz o un conjunto de matrices como entrada, y sobre sus celdas se aplican operaciones matemáticas (productos escalares); además el filtro o kernel, también expresado de forma matricial, trabaja de forma análoga que los pesos de una NN, extrayendo características o *feature maps*; a mayor cantidad de filtros o kernels, se extraerán más características relevantes, lo que supone un aumento del coste computacional. Finalmente, una capa de convolución se puede describir como un conjunto de mapas de activación o *features maps* apiladas que definen la profundidad de la capa (*stacking*); cada celda de la matriz resultante solo cubre una pequeña porción espacial de la capa o imagen anterior, lo que se conoce como campo receptivo. Las capas convolucionales son covariantes a la traslación, pero su geometría 2D es fija, por lo que no son invariantes frente a la rotación, el escalado, la deformación, etc.; es necesario entrenar el modelo para aprender esas invariancias.

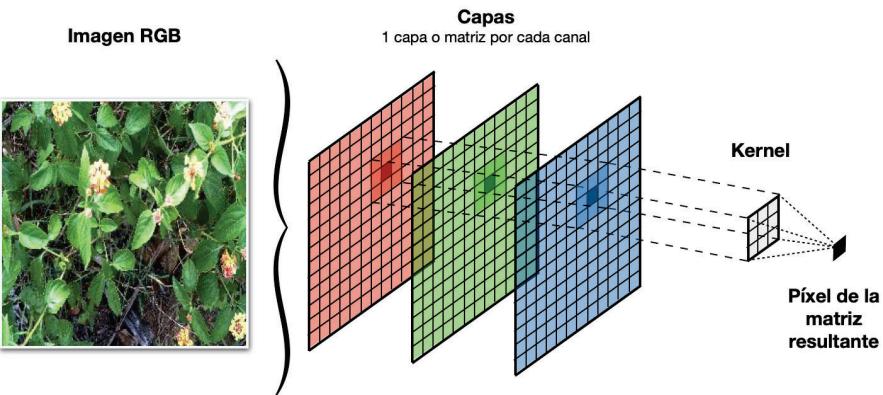


Figura 3.9: Ejemplo de aplicar un kernel sobre una imagen RGB

$  \begin{array}{ c c c c c } \hline  1 & 0 & 0 & 1 & 1 \\ \hline  1 & 0 & 1 & 0 & 1 \\ \hline  1 & 1 & 0 & 0 & 1 \\ \hline  0 & 0 & 1 & 0 & 1 \\ \hline  1 & 0 & 1 & 1 & 1 \\ \hline  \end{array}  \star  \begin{array}{ c c } \hline  1 & 0 & 1 \\ \hline  0 & 1 & 0 \\ \hline  1 & 0 & 1 \\ \hline  \end{array}  = \begin{array}{ c c } \hline  2 & \\ \hline  & \\ \hline  \end{array}  $ $(1*1) + (0*0) + (0*1) + (1*0) + (0*1) + (1*1) + (1*0) + (0*1)$	$  \begin{array}{ c c c c } \hline  1 & 0 & 0 & 1 & 1 \\ \hline  1 & 0 & 1 & 0 & 1 \\ \hline  1 & 1 & 0 & 0 & 1 \\ \hline  0 & 0 & 1 & 0 & 1 \\ \hline  1 & 0 & 1 & 1 & 1 \\ \hline  \end{array}  \star  \begin{array}{ c c } \hline  1 & 0 & 1 \\ \hline  0 & 1 & 0 \\ \hline  1 & 0 & 1 \\ \hline  \end{array}  = \begin{array}{ c c c } \hline  2 & 3 & 3 \\ \hline  4 & & \\ \hline  & & \\ \hline  \end{array}  $ $(1*1) + (0*0) + (1*1) + (1*0) + (1*1) + (0*0) + (0*1) + (1*0)$	$  \begin{array}{ c c c c c } \hline  1 & 0 & 0 & 1 & 1 \\ \hline  1 & 0 & 1 & 0 & 1 \\ \hline  1 & 1 & 0 & 0 & 1 \\ \hline  0 & 0 & 1 & 0 & 1 \\ \hline  1 & 0 & 1 & 1 & 1 \\ \hline  \end{array}  \star  \begin{array}{ c c } \hline  1 & 0 & 1 \\ \hline  0 & 1 & 0 \\ \hline  1 & 0 & 1 \\ \hline  \end{array}  = \begin{array}{ c c c } \hline  2 & 3 & 3 \\ \hline  4 & 0 & 4 \\ \hline  3 & & \\ \hline  \end{array}  $ $(1*1) + (1*0) + (0*1) + (0*0) + (1*1) + (1*0) + (1*1) + (0*0) + (1*1)$
$  \begin{array}{ c c c c c } \hline  1 & 0 & 0 & 1 & 1 \\ \hline  1 & 0 & 1 & 0 & 1 \\ \hline  1 & 1 & 0 & 0 & 1 \\ \hline  0 & 0 & 1 & 0 & 1 \\ \hline  1 & 0 & 1 & 1 & 1 \\ \hline  \end{array}  \star  \begin{array}{ c c } \hline  1 & 0 & 1 \\ \hline  0 & 1 & 0 \\ \hline  1 & 0 & 1 \\ \hline  \end{array}  = \begin{array}{ c c } \hline  2 & 3 \\ \hline  & \\ \hline  \end{array}  $ $(0*1) + (0*0) + (1*1) + (0*0) + (1*1) + (0*0) + (0*1) + (0*1)$	$  \begin{array}{ c c c c } \hline  1 & 0 & 0 & 1 & 1 \\ \hline  1 & 0 & 1 & 0 & 1 \\ \hline  1 & 1 & 0 & 0 & 1 \\ \hline  0 & 0 & 1 & 0 & 1 \\ \hline  1 & 0 & 1 & 1 & 1 \\ \hline  \end{array}  \star  \begin{array}{ c c } \hline  1 & 0 & 1 \\ \hline  0 & 1 & 0 \\ \hline  1 & 0 & 1 \\ \hline  \end{array}  = \begin{array}{ c c c } \hline  2 & 3 & 3 \\ \hline  4 & 0 & 4 \\ \hline  3 & 3 & \\ \hline  \end{array}  $ $(0*1) + (1*0) + (0*1) + (0*0) + (1*1) + (0*0) + (1*1) + (0*0) + (1*1)$	$  \begin{array}{ c c c c c } \hline  1 & 0 & 0 & 1 & 1 \\ \hline  1 & 0 & 1 & 0 & 1 \\ \hline  1 & 1 & 0 & 0 & 1 \\ \hline  0 & 0 & 1 & 0 & 1 \\ \hline  1 & 0 & 1 & 1 & 1 \\ \hline  \end{array}  \star  \begin{array}{ c c } \hline  1 & 0 & 1 \\ \hline  0 & 1 & 0 \\ \hline  1 & 0 & 1 \\ \hline  \end{array}  = \begin{array}{ c c c } \hline  2 & 3 & 3 \\ \hline  4 & 0 & 4 \\ \hline  3 & 3 & 3 \\ \hline  \end{array}  $ $(0*1) + (0*0) + (1*1) + (0*0) + (1*1) + (0*0) + (0*1) + (1*0) + (1*1)$
$  \begin{array}{ c c c c c } \hline  1 & 0 & 0 & 1 & 1 \\ \hline  1 & 0 & 1 & 0 & 1 \\ \hline  1 & 1 & 0 & 0 & 1 \\ \hline  0 & 0 & 1 & 0 & 1 \\ \hline  1 & 0 & 1 & 1 & 1 \\ \hline  \end{array}  \star  \begin{array}{ c c } \hline  1 & 0 & 1 \\ \hline  0 & 1 & 0 \\ \hline  1 & 0 & 1 \\ \hline  \end{array}  = \begin{array}{ c c } \hline  2 & 3 \\ \hline  & \\ \hline  \end{array}  $ $(0*1) + (1*0) + (1*1) + (0*0) + (1*1) + (0*0) + (1*0) + (1*1)$	$  \begin{array}{ c c c c } \hline  1 & 0 & 0 & 1 & 1 \\ \hline  1 & 0 & 1 & 0 & 1 \\ \hline  1 & 1 & 0 & 0 & 1 \\ \hline  0 & 0 & 1 & 0 & 1 \\ \hline  1 & 0 & 1 & 1 & 1 \\ \hline  \end{array}  \star  \begin{array}{ c c } \hline  1 & 0 & 1 \\ \hline  0 & 1 & 0 \\ \hline  1 & 0 & 1 \\ \hline  \end{array}  = \begin{array}{ c c c } \hline  2 & 3 & 3 \\ \hline  4 & 0 & 4 \\ \hline  3 & 3 & 3 \\ \hline  \end{array}  $ $(0*1) + (1*0) + (0*1) + (0*0) + (1*1) + (0*0) + (1*1) + (0*0) + (1*1)$	$  \begin{array}{ c c c c c } \hline  1 & 0 & 0 & 1 & 1 \\ \hline  1 & 0 & 1 & 0 & 1 \\ \hline  1 & 1 & 0 & 0 & 1 \\ \hline  0 & 0 & 1 & 0 & 1 \\ \hline  1 & 0 & 1 & 1 & 1 \\ \hline  \end{array}  \star  \begin{array}{ c c } \hline  1 & 0 & 1 \\ \hline  0 & 1 & 0 \\ \hline  1 & 0 & 1 \\ \hline  \end{array}  = \begin{array}{ c c c } \hline  2 & 3 & 3 \\ \hline  4 & 0 & 4 \\ \hline  3 & 3 & 3 \\ \hline  \end{array}  $ $(0*1) + (0*0) + (1*1) + (0*0) + (1*1) + (0*0) + (0*1) + (1*0) + (1*1)$

Figura 3.10: Ejemplo de aplicar un kernel o filtro 3x3 sobre una capa de entrada 5x5 con *stride* 1

Ligado al concepto del kernel y las operaciones convolucionales, existen diferentes formas de aplicar estos filtros. Como se aprecia en la Fig. 3.10, se introduce el concepto de *stride* o paso, que hace referencia a la forma de aplicar y mover el kernel sobre el *feature map* de entrada que recibe la capa convolucional. Como se observa en el ejemplo gráfico, el kernel recorre la imagen desplazándose *s* píxeles hacia la derecha o hacia abajo durante cada iteración, esto es lo que se entiende por salto, pudiendo modificar este número de píxeles a desplazar dependiendo del problema; es posible realizar la convolución con un *stride* diferente e incluso mayor, lo que permite reducir el tamaño de la imagen (*subsampling*), en comparación con el tamaño original de la entrada. A continuación se muestra un ejemplo con un *stride* = 2 (Fig. 3.11).

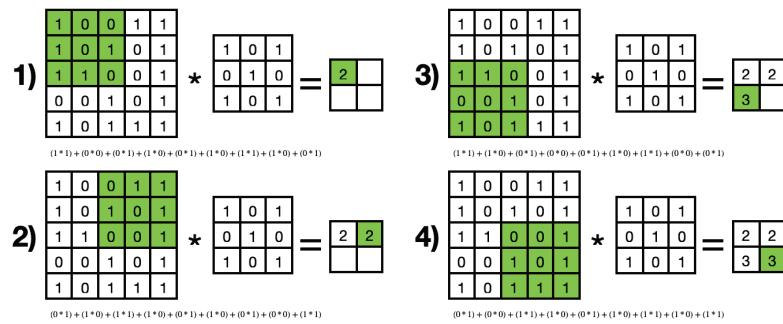


Figura 3.11: Ejemplo de aplicar un kernel o filtro  $3 \times 3$  sobre una capa de entrada  $5 \times 5$  con  $stride = 2$

Como se puede intuir con estos ejemplos, pueden llegar a aparecer incompatibilidades con los tamaños y dimensiones de las matrices, una forma de solucionar estas discrepancias puede ser incluir una fila de celdas “extra.” en los bordes exteriores de la matriz, aumentando así su tamaño en  $n$  celdas; esta técnica se conoce como *padding*, consiste en incluir celdas a una matriz determinada hasta que tenga las dimensiones deseadas, estas celdas pueden tomar diferentes valores, desde los mismos que las celdas contiguas, hasta valores constantes como 1 o 0 (*zero-padding*), como se aprecia en la Fig. 3.12.

0	0	0	0	0	0	0
0	1	0	0	1	1	0
0	1	0	1	0	1	0
0	1	1	0	0	1	0
0	0	0	1	0	1	0
0	1	0	1	1	1	0
0	0	0	0	0	0	0

Figura 3.12: Ejemplo de *zero-padding* sobre una matriz  $5 \times 5$ , obteniendo finalmente una matriz  $7 \times 7$

Muy relacionado con estos conceptos ya presentados, se encuentra la técnica de submuestreo, *subsampling* o *downsampling*, conocida como *pooling*, normalmente se aplica entre capas convolucionales, exactamente sobre el resultado de haber aplicado una función de activación tras las operaciones convolucionales; es una operación que analiza el contenido de una imagen por regiones o bloques, para así extraer la información más representativa. Su principal objetivo es reducir el tamaño de la imagen, para reducir la dificultad computacional; misma situación que la comentada anteriormente al aplicar un  $stride = 2$ . Para ello, resume los valores de la vecindad de un píxel; introduce invariancia ante transiciones locales, reduce el número de unidades ocultas y descarta información sobre la ubicación de características en la imagen. Generalmente, no hay solapamiento durante la aplicación, la ventana suele tener unas dimensiones reducidas y existen diferentes tipos de operaciones, como el cálculo del máximo, el mínimo o el valor medio de la ventana (Fig. 3.13).

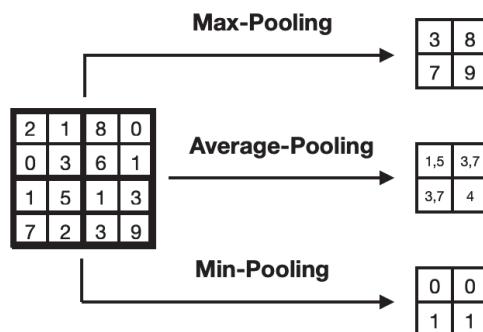


Figura 3.13: Ejemplo de tres tipos de *pooling*; *max*, *min* y *average pooling*

Dentro del marco de extracción de características, cabe destacar el uso generalizado de la función de activación ReLU, anteriormente explicada en Sec. 3.1.2.1. La razón principal por la que se utiliza es para aumentar la no linealidad en las imágenes; las imágenes tienen una naturaleza lineal, aunque también muestran elementos no lineales como las transiciones entre píxeles o bordes; aumentando las diferencias entre píxeles vecino, consiguiendo progresiones más abruptas y definidas. También aparece el concepto de *batch normalization* [198], inicialmente presentado para reducir el *Internal Covariate Shift* [199] (Fig. 3.14). La normalización en lotes consiste en añadir un paso extra entre los *feature maps* y la función de activación para normalizar las salidas y que estas estén entre 0 y 1.

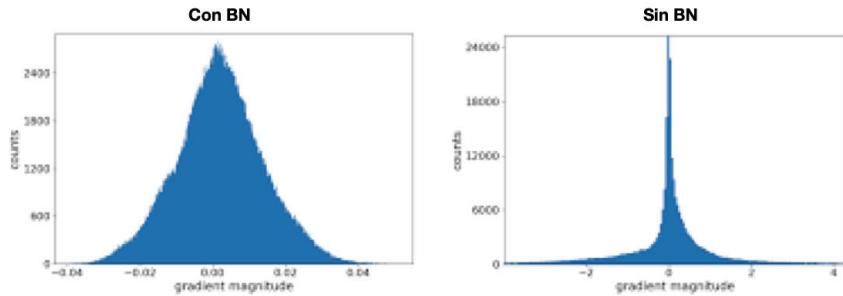


Figura 3.14: Efecto sobre la pérdida al aplicar la técnica de *Batch Normalization*

Por último, para completar y finalizar el proceso de clasificación, se encuentran las capas de salida, también denominadas capas totalmente conectadas o *Fully Connected Layers*. Estas se sitúan al final de la arquitectura de la red, precedidas por una capa *flatten layer* que busca convertir los *feature maps* en vectores unidimensionales, es decir, toma la salida de las capas anteriores, matrices bidimensionales, y las “aplana” o “estira” convirtiéndolas en un único vector. Este vector unidimensional servirá de entrada para las capas totalmente conectadas (Fig. 3.15).

Para reemplazar las capas tradicionales densas o totalmente conectadas, se proponen las capas de promedio global (*global average layer*); se aplica al final de la red, tras una capa convolucional, y la idea principal es generar un *feature map* por cada categoría o etiqueta de clasificación sustituyendo así a las FC (Fig. 3.15). En este proyecto se fija el bloque final de clasificación, sustituyendo la capa *flatten* y densa, por una capa *global average pooling* + una capa densa.

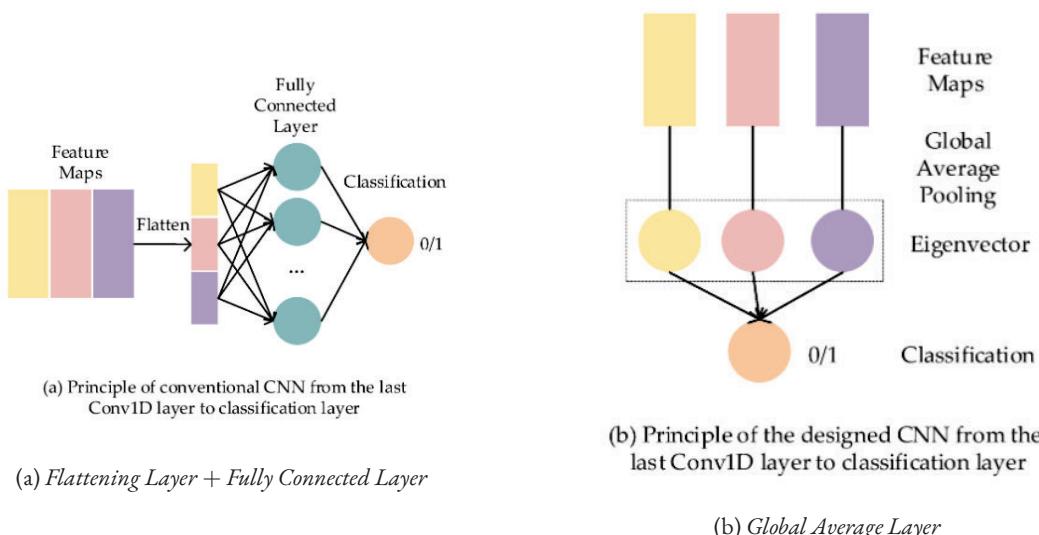


Figura 3.15: Ejemplo del funcionamiento de las capas totalmente conectadas y las capas de promedio global [7]

**Arquitecturas clásicas** A lo largo de los años, y con el avance de la tecnología, se han desarrollado diferentes arquitecturas que han destacado en diversas competiciones y retos de clasificación de imágenes como son ImageNet, *Large Scale Visual Recognition Challenge* (ILSVRC) [200] o International Conference on Pattern Recognition (ICPR) [201]. Estas arquitecturas han sido aplicadas y modificadas para numerosos y diversos retos; a continuación se explican en detalle algunas de las arquitecturas más importantes y relevantes para este proyecto en particular.

- **VGGNet** [202]: la red de Karen Simonyan y Andrew Zisserman fue el subcampeón en el ILSVRC del 2014. Su principal contribución fue mostrar que la profundidad de la red es un componente crítico para un buen desempeño. La VGGNet tiene diferentes versiones, por ejemplo la VGG16 contiene 16 capas CONV/FC y presenta una arquitectura extremadamente homogénea; solo realiza convoluciones  $3 \times 3$  y agrupación  $2 \times 2$  desde el principio hasta el final. Es más caro de evaluar y utiliza mucha más memoria y parámetros. La mayoría de estos parámetros se encuentran en la primera capa de *fully connected* y se descubrió que estas capas se pueden eliminar sin disminuir el rendimiento, reduciendo así la cantidad de parámetros necesarios (Fig. 3.16).

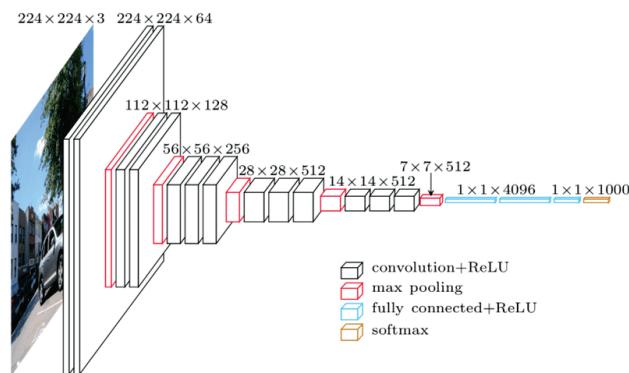


Figura 3.16: Arquitectura de la VGG16. Imagen de [8]

- **GoogleNet (Inception v1)** [203]: ganadora del reto ILSVRC 2014. Fue desarrollada por Google, pero sus autores hacen tributo en el propio nombre a la arquitectura LeNet [204], en la cual se basaron para su desarrollo. La arquitectura de esta red es bastante compleja en comparación con las existentes hasta el momento, ya que fue un modelo bastante diferente al general que había sido presentado hasta entonces. En esta se disponen varios bloques, donde se realizan diversas operaciones en paralelo, en vez de en forma secuencial como se había estado haciendo hasta este momento. En la Figura 3.17, se puede ver como es la arquitectura de esta red y los bloques que la conforman.

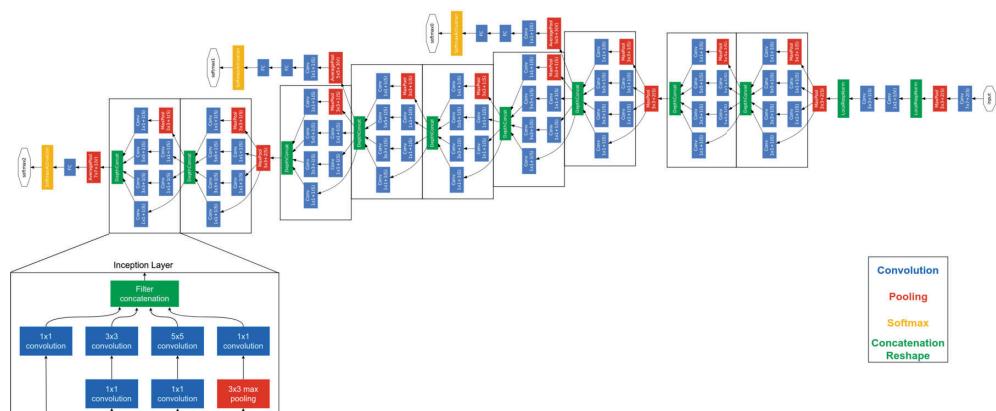


Figura 3.17: Arquitectura de GoogleNet. Imagen de [?]

- **ResNet** [9]: desarrollado por Kaiming He et al., fue el ganador del ILSVRC 2015. Cuenta con conexiones de salto especiales y hace un uso intensivo de la normalización por lotes; debido a los bloques residuales, se resuelve el problema de la desaparición del gradiente (*Vanishing gradient*) y el modelo es capaz de retener las capas útiles y no usar las que no ayudan. La arquitectura no cuenta con capas totalmente conectadas (*Fully connected layer*) al final de la red. El objetivo de los bloques residuales es introducir cierta variación en la imagen de salida, conservando el procesado, los detalles de la imagen de entrada y su contexto; en las arquitecturas lineales, este contexto se desvanece con la profundidad de la red (Fig. 3.18).

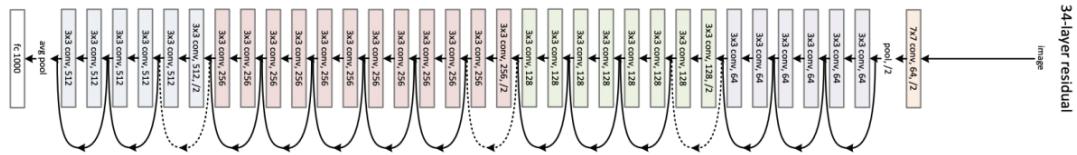


Figura 3.18: Arquitectura de la ResNet. Imagen de [9]

## 3.2. Construcción, diseño y optimización de redes

Una vez diseñada la arquitectura de la red, se debe comenzar con el entrenamiento, lo que se entiende como el proceso de aprendizaje de las redes neuronales; este proceso consiste en presentar el problema a la red, y que esta extraiga y aprenda de manera progresiva aquellas características más importantes para la resolución final del problema. Por lo que, básicamente, es o se pueden entender como un problema de optimización.

### 3.2.1. Entrenamiento

Entrenar una NN consiste en ajustar la red, de tal forma que las respuestas de la capa de salida se parezcan o se ajusten lo máximo posible a los datos ya conocidos. Para ello, se parte de un conjunto de datos etiquetados donde un porcentaje de estos servirán para alimentar la red durante el entrenamiento, es decir, serán la primera entrada o *input* que recibirá el bloque de entrenamiento; y el resto de datos etiquetados se utilizarán para los procesos de validación y evaluación.

Durante el proceso de entrenamiento se observa la diferencia entre las predicciones y el *target* u objetivo, viendo así qué tan similar es el resultado obtenido con respecto a su valor real; este valor se conoce como pérdida (*loss*). Se busca minimizar la pérdida ajustando los pesos a lo largo del proceso de entrenamiento. Los pesos son un conjunto de números que almacenan las especificaciones de las capas; el propósito final es encontrar los mejores valores para estos parámetros y así obtener la mejor relación *input-output*. Debido a la complejidad de las redes y su alto número de parámetros, encontrar el valor correcto para los pesos es una ardua tarea, ya que la modificación del valor de un parámetro afecta al comportamiento de los demás. Para ver la desviación entre la salida de la red y la solución del problema, existen diferentes técnicas de cálculo del error, destacando: el Error Cuadrático Medio (ECM), el Error Medio Absoluto (EAM), y la Entropía Cruzada [205]; para optimizar el problema, se busca que este error sea nulo.

La red utiliza este error como retroalimentación para recortar la distancia entre las predicciones y las etiquetas reales, y como consecuencia poder actualizar el valor de los pesos. Este ajuste es obra del optimizador y algoritmo de retropropagación o backpropagation [206]. Inicialmente, los pesos de la red se asignan aleatoriamente, lo que implica que el valor de pérdida sea muy elevado durante las primeras *epochs*. Durante los distintos bucles de entrenamiento, se aplican transformaciones sobre los pesos, enfocadas en la disminución del valor de pérdida hasta alcanzar la convergencia del algoritmo de aprendizaje en una aproximación del error aceptable, próximo a 0. Logrando el propósito final, la similitud o cercanía entre los objetivos reales y las salidas de la red entrenada. Este proceso de entrenamiento básico y general de una red, se representa gráficamente en la Fig. 3.19.

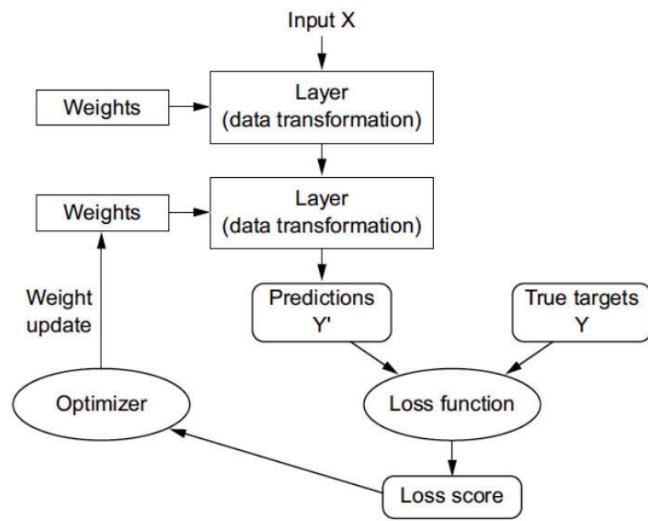


Figura 3.19: Entrenamiento de una NN básica: Pesos, Función de pérdida, Optimizador y Retropropagación

### 3.2.1.1. Técnicas de optimización y regularización

**Optimización** Se define como la búsqueda o llegada a una situación óptima, es decir, la obtención de los mejores resultados posibles. El entrenamiento de una red puede entenderse como un proceso de optimización, ya que se busca obtener los mejores resultados posibles a partir de un proceso iterativo que se encuentra en constante cambio, ajustándose en la dirección más óptima; lo que se puede entender como el proceso conjunto con el algoritmo de retroalimentación o *backpropagation*, en el que se busca que el valor de la pérdida o *loss* sea mínima o muy próxima a cero (Fig. 3.19). El optimizador, es el encargado de generar pesos mejores, y su aplicación en el proceso de entrenamiento es crucial.

Muy relacionado con los optimizadores, aparece el concepto del ratio de aprendizaje. El *learning rate* o ratio de aprendizaje indica el tamaño del salto en cada una de las iteraciones, por lo que regula la velocidad de optimización. La definición de este hiperparámetro es relativamente compleja, ya que si toma un valor muy pequeño, el proceso de actualización se encarece, necesitando mucho tiempo para converger, y puede quedarse atrapado en mínimos locales; mientras que si el valor es muy alto, aunque en términos de velocidad este sea superior, el proceso de actualización puede sobrepasar el mínimo global, como se representa en la Fig. 3.20. El valor del ratio de aprendizaje tiene un efecto directo en el proceso de entrenamiento (Fig. 3.21).

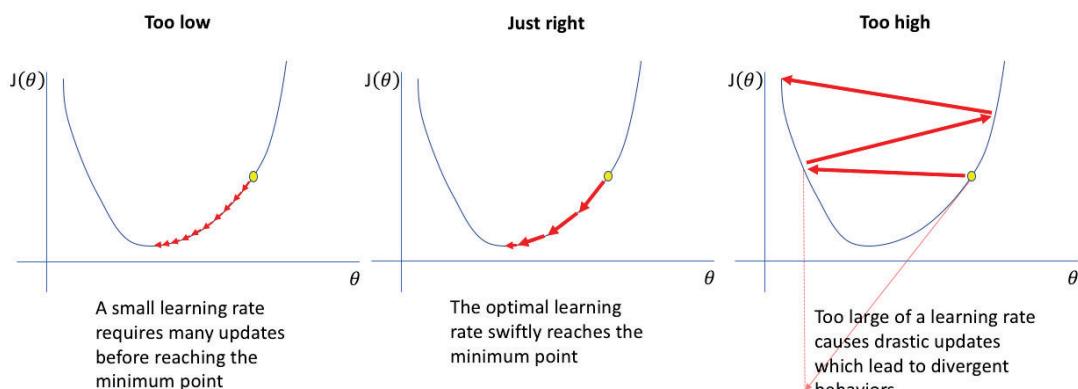


Figura 3.20: Ratio de aprendizaje o Learning rate. Imagen de [\[10\]](#)

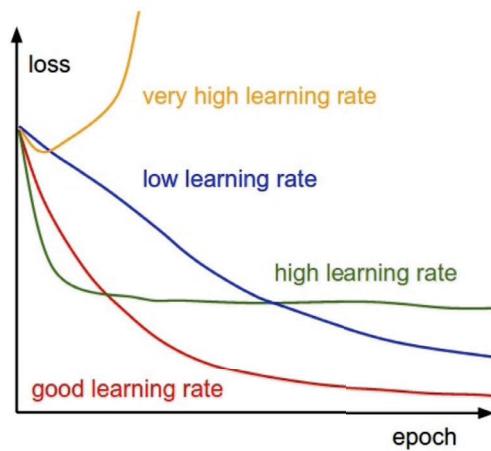


Figura 3.21: Efecto en el entrenamiento del valor del ratio de aprendizaje definido

Existen numerosos algoritmos de optimización [207]:

- **Descenso estocástico del gradiente (Stochastic Gradient Descent, SGD):** este método de optimización consiste en desplazarse en dirección contraria al gradiente según un ratio de aprendizaje; se añade cierto grado de oscilación para incrementar la exploración y evitar quedarse atrapado en mínimos locales.
- **Momentum:** este optimizador crea un movimiento con inercia que promueve y fomenta la convergencia. Con este optimizador, los pasos o saltos son irregulares y acelera el descenso, hace que disminuyan las oscilaciones hacia la dirección del gradiente lo que le permite evitar mínimos locales .
- **Gradiente acelerado de Nesterov (Nesterov accelerated gradient, NAG):** tanto este optimizador como el Momentum, se pueden considerar extensiones del SGD. Aunque el Momentum acelera el proceso de convergencia y reduce el riesgo de quedar atrapado en mínimos locales, también tiende a sobrepasar los valores (*overshooting*). Este optimizador busca superar estos inconvenientes actuando como un método en dos pasos, prediciendo y corrigiendo. La parte predictora se encarga de extrapolar la trayectoria en un punto; una vez esté en el punto predicho, se realiza una corrección de la trayectoria, de forma similar al mecanismo de inercia utilizado en el Momentum, manteniendo así todas las ventajas de este método.

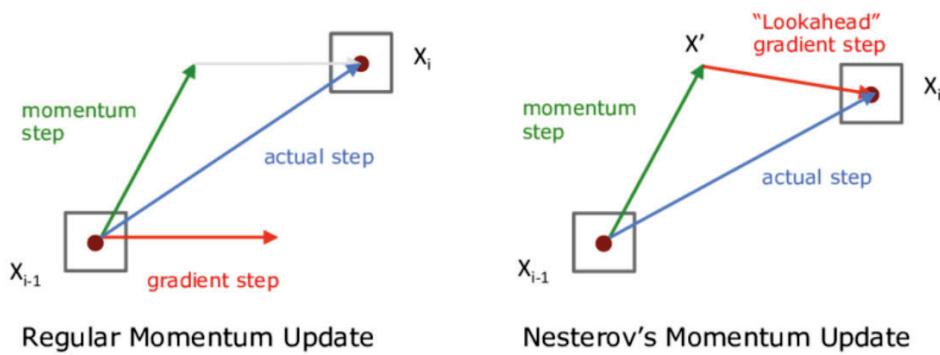


Figura 3.22: Actualización del gradiente con el optimizador Momentum (izq.) y con Nesterov (dch.) [11]

- **AdaGrad:** este método se enfoca en estimar mejor la dirección de descenso y se caracteriza por tener un *learning rate* adaptable a los parámetros de la red; aumentando su tamaño con parámetros que apenas se modifican, mientras que este ratio toma valores más pequeños con aquellos parámetros que se modifican con frecuencia. Esta adaptación del *lr*, aumenta la robustez de la búsqueda.

- **RMSProp**: es un algoritmo similar al AdaGrad, también mantiene un factor de entrenamiento diferente para cada dimensión pero en este caso, en vez de tener acumulados todos los gradientes únicamente considera los  $w$  gradientes más recientes. RMSProp también reduce el paso en la dirección de gradientes más grandes, solucionando los problemas que da su antecesor, ya que con AdaGrad a medida que avanza el tiempo el  $lr$  se hace demasiado pequeño, ralentizando el proceso de ajuste.
- **Adam**: es una combinación entre RMSProp y Momentum. calcula una combinación lineal entre el gradiente y el incremento anterior (Momentum); además, considera los gradientes más recientes para mantener diferentes  $lr$  en función de los parámetros de la red (RMSProp).

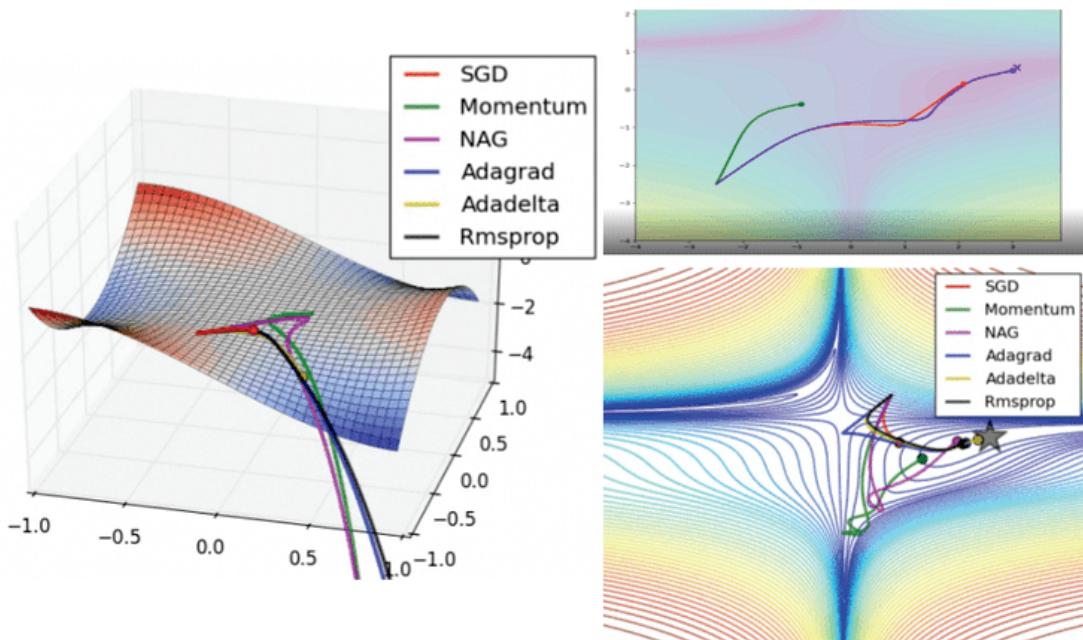


Figura 3.23: Ejemplo gráfico de las diferencias de actuación de algunos de los optimizadores más populares

**Regularización** Durante el proceso de entrenamiento, se busca minimizar al máximo el error o pérdida, para ello se diseñan redes con alta capacidad de clasificación. Hoy en día, la tecnología es muy potente y accesible, facilitando los entrenamientos; por eso mismo, gracias a los potentes recursos disponibles, se ha tendido a aumentar la complejidad de las redes para mejorar su rendimiento y capacidad predictora; con este aumento excesivo del número de capas y parámetros de la red, se introduce el concepto del sobreajuste u *overfitting* [208]. El sobreajuste de una red consiste en que esta “sobreaprenda” durante el proceso de entrenamiento, perdiendo así su capacidad de generalizar (Fig. 3.24).

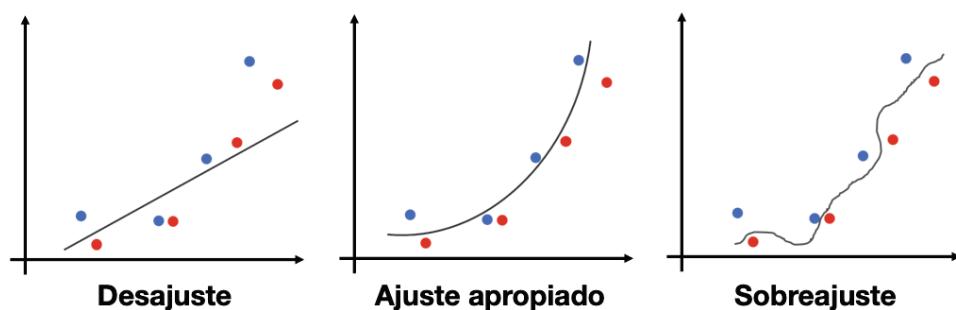


Figura 3.24: Ejemplo gráfico de los posibles ajustes a realizar, incluido el sobreajuste u *overfitting*

Por eso mismo, se busca una compensación o equilibrio entre el sesgo y la varianza (*bias-variance tradeoff*, Fig. 3.25)[209, 210]; siendo el sesgo o varianza, las diferencias y cambios que sufre la capacidad de ajuste del modelo con conjuntos de datos diferentes, siendo muy grande si el modelo tiene tanta capacidad de ajuste que termina aprendiendo el ruido (*overfitting*); y entendiendo el *bias* como la distancia del modelo medio a la solución, es decir, la diferencia entre las soluciones y los resultados o *outputs* devueltos por el modelo entrenado, esta es grande si el modelo tiene muy poca capacidad de clasificación o ajuste de los datos. El *overfitting* se puede dar por diferentes motivos, como un entrenamiento con pocas muestras o datos, o el uso de arquitecturas demasiado complejas para el problema a resolver, haciendo que la red pueda llegar a memorizar los datos y las soluciones, obteniendo métricas de rendimiento muy altas durante el entrenamiento pero valores muy deficientes al evaluar el modelo con imágenes desconocidas (Fig. 3.25).

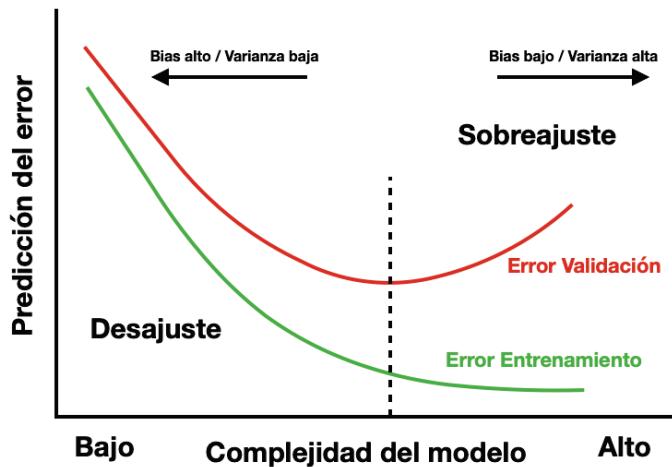


Figura 3.25: Ejemplo de *overfitting* y *underfitting* durante el proceso de entrenamiento

Por eso mismo, para combatir el sobreajuste aparecen los métodos de regularización que añaden una penalización sobre la función de coste, es decir, reducen los grados de libertad del modelo para evitar o contrarrestar el sobreajuste, y así conseguir que el modelo resultante sea más simple y generalice adecuadamente, es decir, sea extrapolable y aplicable a otros conjuntos de datos o imágenes ajenas al *dataset* original. Algunos de los métodos de regularización más utilizados son [211]:

- **Regularización L1 y L2:** buscan reducir el valor de los parámetros. Ambos métodos introducen un término adicional de penalización en la función de coste original L, valor al que se le añade la suma de los pesos ( $L2: \sum |w|^2$ ;  $L1: \sum w$ ). Este sumatorio de los pesos puede alcanzar valores muy altos, minimizando extremadamente la función de coste, aproximando los parámetros a 0; por eso mismo se incluye una constante pequeña  $\lambda$ , escogida arbitrariamente.
- **Dropout:** este método consiste en eliminar aleatoriamente un porcentaje de neuronas durante el proceso de entrenamiento, es decir, se activan y desactivan ciertas neuronas en función de una probabilidad previamente definida; esta probabilidad puede ser constante o particular para cada capa.
- **Early stopping:** técnica utilizada para saber cuando se debe parar el entrenamiento, antes de producir un sobreajuste. Basándose en el concepto de *learning rate* adaptativo (Fig. ??), se introduce la técnica de parada temprana o *early stopping*; ambas técnicas tienen un comportamiento similar. Se escoge una variable y se determina un rango o valor de paciencia, si esta variable no disminuye a medida que avanza el entrenamiento y se sobrepasa el valor de paciencia definido sin que la variable haya sufrido ninguna mejora, el entrenamiento finaliza.
- **Transferencia de aprendizaje:** es una de las técnicas más importantes dentro del DL. Parte de la idea de que no es necesario un conjunto de datos muy grande sobre una temática concreta para entrenar una

red adaptada a un problema determinado, es decir, que no es necesario entrenar los modelos desde cero sino que se puede partir de una red ya entrenada (pre-entrenada). Esta técnica se basa en la capacidad de generalización de las arquitecturas pre-entrenadas, para poder reutilizar sus características y parámetros ya ajustados como punto de partida para el entrenamiento de otro modelo, con unos datos más reducidos y específicos para una tarea determinada, a modo de especialización.

- **Data augmentation:** consiste en aplicar diversas transformaciones sobre las entradas originales, obteniendo muestras ligeramente diferentes pero sin modificar la información original, favoreciendo el proceso de entrenamiento y facilitando la capacidad de generalización del modelo. Técnica de regularización muy utilizada en el campo de la visión artificial, donde los datos originales son imágenes digitales; se busca que la red no pueda memorizar la imagen, por eso mismo se aplican estas transformaciones de forma aleatoria. Existen diferentes tipos de transformaciones, como la rotación, recorte, cambio de escala, cambio en los niveles de brillo, aplicación de ruido, etc.

### 3.3. Computación Evolutiva

La computación evolutiva es un conjunto de técnicas de optimización bioinspiradas, utilizadas para resolver problemas de búsqueda y optimización. Estos métodos simulan poblaciones de individuos que codifican posibles soluciones a un problema, y evolucionan con el objetivo de mejorar y llegar a la solución óptima. Los enfoques evolutivos imitan el proceso natural de selección, la adaptación al medio y la supervivencia del más fuerte, es decir, los individuos más aptos. El objetivo es llegar a una solución óptima modificando la población y los individuos que la componen, haciendo que estos evolucionen para finalmente converger en el óptimo global. Sin embargo, partiendo del teorema “*no-free-lunch*” (NFL) [212], no existe ningún algoritmo que sea capaz de resolver todos los problemas; por tanto, no hay ningún algoritmo que destaque frente al resto, aunque sí existen algoritmos evolutivos cuyo rendimiento sea notablemente superior en determinados problemas o retos. Algunos de los enfoques evolutivos más populares son:

- **Programación Evolutiva (EP)** [213]: se centra en la evolución de varios parámetros de programas informáticos fijos. Hace uso de los operadores de selección y mutación aleatoria, pero no de cruce. El tipo de codificación utilizada no es de gran importancia, no se ve afectado por esta elección.
- **Algoritmos Genéticos (GA)** [214]: se caracteriza por realizar una selección probabilística y definir una probabilidad de mutación muy baja, siendo el operador de cruce el principal. Generalmente, utiliza la codificación binaria (0 y 1) (Fig. 3.27), aunque también es posible aplicar otras. Para este proyecto se pone en práctica un algoritmo evolutivo que cumple con todos los requisitos mencionados.
- **Estrategias Evolutivas (ES)** [215]: usa representaciones independientes del problema natural; su codificación hace uso de números naturales que dependiendo del problema, pueden ser representados de forma continua o discreta (Fig. 3.27). Hace uso del operador genético de selección y mutación, siendo la probabilidad de este último muy alta.
- **Programación genética (GP)** [216]: busca construir programas de forma autónoma, sin haber sido diseñados previamente por un humano. La codificación utilizada representa a los individuos con estructuras complejas en forma de árbol. Los árboles están compuestos por nodos que pueden representar símbolos terminales (T) o no terminales (NT) (Fig. 3.27); esta codificación puede generar expresiones de forma fácil y rápida.

#### 3.3.1. Características y funcionamiento

Como se ha comentado, la computación evolutiva está compuesta por diversos algoritmos que comparten tanto la influencia darwiniana, que persigue la supervivencia del más fuerte, como las etapas que constituyen el flujo o ciclo evolutivo de estos algoritmos, descrito y representado gráficamente a continuación (Fig. 3.26).

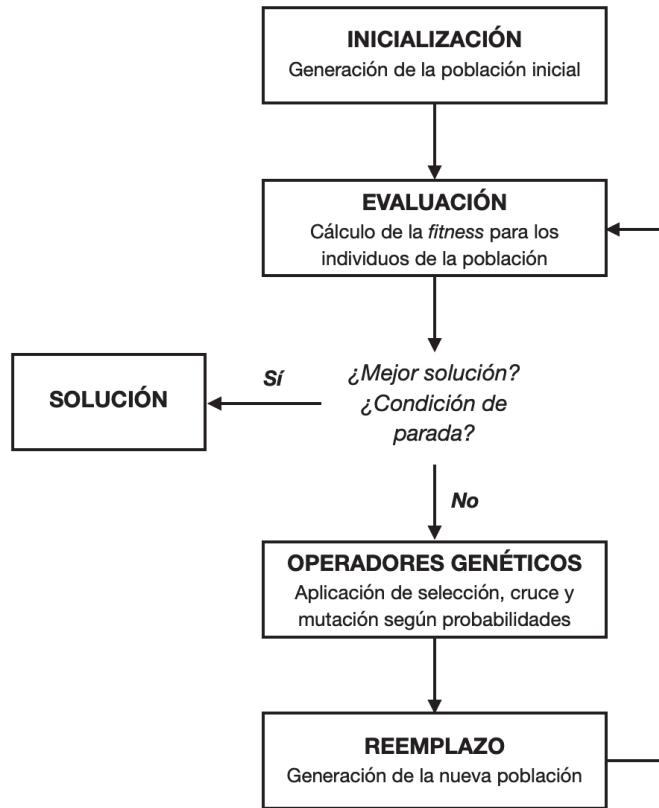


Figura 3.26: Ciclo evolutivo genérico de los Algoritmos Evolutivos

- **Inicialización:** los algoritmos genéticos comienzan con lo que se conoce como inicialización, donde del espacio de búsqueda se seleccionan  $N$  posibles soluciones para un problema de optimización; para ello, se generan y codifican las soluciones de forma totalmente aleatoria. La generación aleatoria, en este caso de una población con un tamaño determinado, también se conoce como inicialización por fuerza bruta [217].
- **Evaluación:** en esta fase se busca medir la capacidad de adaptación de un individuo al problema, es decir, que devuelve un valor cuantificable que expresa cómo de buena o mala es una solución. Los individuos que constituyen la población se encuentran codificados (Sec. 3.3.1.1), y es sobre estas codificaciones donde se efectuará el proceso de reproducción, pero la evaluación se realiza sobre las soluciones, por lo que los individuos se deben decodificar. La elección de una función de evaluación y la codificación utilizada, son dos aspectos clave para el éxito y correcto desarrollo del ciclo evolutivo; la función objetivo debe ser diseñada para cada problema particular, es decir, no existe una función de evaluación genérica.
- **Operadores genéticos:** están inspirados en los mecanismos evolutivos naturales, y engloba a todas las fases del proceso de reproducción: selección, cruce y mutación. Primero, se seleccionan algunos individuos de la población para tener descendencia, conocidos como individuos progenitores. Para generar los descendientes, se aplicarán o no los operadores de cruce y mutación, dependiendo de la probabilidad definida para cada uno de estos operadores y del nivel de adaptación de cada uno de los progenitores. Esta fase del proceso evolutivo se encuentra explicada en más detalle en la Sec. 3.3.1.2.
- **Reemplazo:** una vez aplicados los operadores genéticos y haber obtenido los descendientes, es necesario formar una nueva población o generación. Esta nueva población debe tener las dimensiones que la original o inicial, y es un fase indispensable dentro del ciclo evolutivo, ya que da paso a un proceso iterativo que puede llegar a quedarse estancado en óptimos locales por una mala elección de técnica, dando

lugar a una falta o ausencia de diversidad genética; cada una de estas iteraciones se denomina generación. Existen diferentes métodos de reemplazo, todos apoyados en el fundamento darwiniano, promoviendo una competición entre los individuos o posibles soluciones del problema por su supervivencia dentro del grupo; descrita con más detalle en la Sec. 3.3.1.3.

- **Fin del ciclo evolutivo:** este proceso iterativo finaliza cuando se ha llegado a la solución deseada o se cumple una condición o criterio de parada (tiempo de procesamiento máximo, número máximo de generaciones, se cubre un rango o índice de satisfacción, falta de diversidad, ausencia de mejora, etc.).

### 3.3.1.1. Codificación

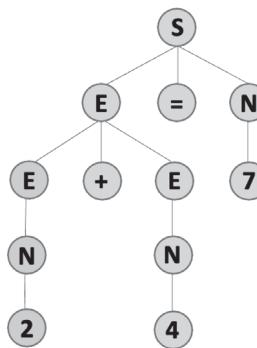
Como se ha comentado anteriormente, se necesita un método de codificación de los puntos del espacio de búsqueda, es decir, una forma de representar las posibles soluciones del problema de forma unificada y general, transformando el dominio de la función en un conjunto acotado de valores representados por cadenas totalmente independientes de su origen. Existen diferentes tipos [178] (Fig. 3.27), la codificación binaria, 0, 1; la codificación no binaria, con número enteros o con números reales; y la **codificación por gramáticas** utilizada en la PG (gramáticas de contexto libre); y la codificación por gramáticas utilizada en la PG (gramáticas de contexto libre), que utiliza árboles para codificar programas de ordenador u otros objetos como formulas lógicas o aritméticas.

0	1	0	1	1	0
---	---	---	---	---	---

(a) Algoritmo genético y codificación binaria

6,0	-0,2	3,9	1,1	-5,9	7,4
-----	------	-----	-----	------	-----

(b) Estrategias evolutivas y codificación no binaria (números reales)



(c) Programación genética y estructura en árbol con codificación por gramáticas

Figura 3.27: Ejemplos de algoritmos evolutivos y su codificación

Se suele utilizar una codificación completa; utilizando un alfabeto  $A$  de cardinal finito  $m$  y cadenas de longitud  $l$ , se pueden formar  $m^l$  posibles cadenas que corresponden a la codificación de puntos únicos dentro del dominio del problema, definiendo así el cardinal del espacio de búsqueda.

Por ejemplo, si nos encontramos en una codificación binaria donde el alfabeto (A) solo se compone de los valores 0 y 1, entonces  $m = 2$ , y siendo el valor máximo de la longitud de las cadenas de 4 bits ( $l = 4$ ), solo es posible generar un máximo de  $2^4 = 16$  combinaciones diferentes y únicas, es decir, solo se podrían representar 16 puntos del espacio de soluciones, como se representa en la Fig. 3.28. Por eso mismo, es tan importante la elección de la codificación (alfabeto, con su respectivo cardinal, y la longitud de las cadenas), ya que definen la longitud del espacio de búsqueda, por lo que para cubrir todo el espacio de búsqueda es necesario definir y elegir correctamente estos parámetros; en el caso de que se quiera aumentar el cardinal del espacio de búsqueda porque este no se cubra el espacio en su totalidad y se pierdan posibles soluciones óptimas para el problema, será necesario aumentar la longitud de las cadenas o bien, elegir un alfabeto de mayor cardinal.

<b>Alfabeto (A) = 0,1</b>	<i>*codificación binaria</i>	1. 0000	9. 1101
<b>Cardinal (m) = 2</b>	<i>*longitud del alfabeto</i>	2. 1111	10. 1011
<b>Longitud de la cadena (l) = 4</b>		3. 0001	11. 1001
<b>Combinaciones posibles = <math>m^l = 2^4 = 16</math></b>		4. 0011	12. 0101
		5. 0111	13. 1010
		6. 1000	14. 0110
		7. 1100	15. 0100
		8. 1110	16. 0010

Figura 3.28: Ejemplo de codificación completa

Este conjunto de puntos del espacio de búsqueda codificados con cadenas que representan posibles soluciones del problema, componen la población con la que trabaja el algoritmo durante el ciclo evolutivo. Basándonos en la representación de la Fig. 3.29, estos individuos están codificados en forma de cadena, lo que se conoce como cromosoma o genoma. A su vez, un cromosoma está compuesto por diferentes genes, que representan una posición de la cadena o cromosoma; dependiendo de la codificación, en concreto del alfabeto elegido y su cardinalidad, los diferentes valores que puede tomar un gen se denominan alelos, es decir, el número de alelos posibles coincide con  $m$  (codificación binaria:  $m = 2 \rightarrow \text{alelos} = 2$ ). También se ha de diferenciar entre el genotipo, que hace referencia al tipo de codificación; y el fenotipo, que es la decodificación del cromosoma, es decir, el valor obtenido al pasar de la representación codificada a la representación de la función objetivo.

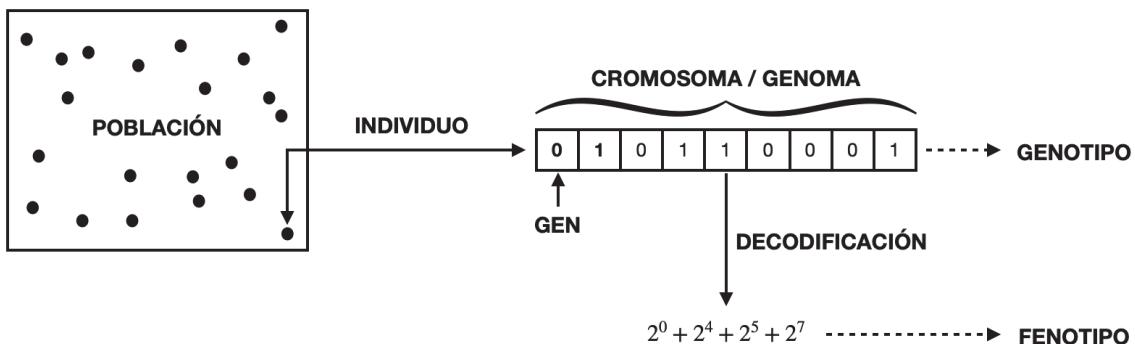


Figura 3.29: Ciclo evolutivo genérico de los Algoritmos Evolutivos

### 3.3.1.2. Operadores genéticos

Los operadores genéticos están inspirados en la evolución natural y se consideran fundamentales para el correcto funcionamiento del ciclo evolutivo; estos imitan el proceso de reproducción natural, generando cambios y promoviendo la adaptación de los individuos al medio, rescatando y manteniendo aquellos cambios que mejoran la capacidad de adaptación. Dentro de estos operadores se encuentran los procesos de selección, cruce y mutación. Estos operadores se utilizan en todos los algoritmos evolutivos, independientemente del enfoque o la codificación; independientemente del tipo de codificación la esencia de los operadores es la misma, es decir, su función y efecto se mantiene, aunque los métodos y formas de aplicación se adaptarán al tipo de codificación utilizada [178]. Por eso mismo, el resumen teórico-práctico se centra en los algoritmos genéticos con codificación binaria ya que ha sido el enfoque elegido para este proyecto.

**Selección** Se encarga de seleccionar de entre todos los individuos de la población aquellos que van a tener descendencia, es decir, hace una selección de los progenitores. Para ello, se duplican los progenitores, manteniéndolos en un lugar intermedio denominado zona de apareamiento o *matting pool*. Aquellos individuos

mejor adaptados tendrán más probabilidades de ser progenitores que los peor adaptados, admitiendo que estos últimos también tengan posibilidades de participar en el proceso reproductivo, aunque con una probabilidad menor; esta distinción enriquece y dirige el proceso evolutivo sin llegar a limitarlo, respetando y favoreciendo la diversidad de la población. Existen diferentes estrategias de selección que pueden ser seguidas en función del problema a resolver, a continuación se explican los métodos más extendidos.

- **Selección aleatoria** (*Random selection*): se toman a los individuos de forma aleatoria, no se tiene en cuenta el nivel de adaptación de los individuos, teniendo todos los individuos la misma probabilidad de ser seleccionados. Es un método simple pero no ideal para todos los problemas, ya que no tiene en cuenta ningún tipo de información sobre la *fitness*.
- **Selección por torneo** (*Tournament Selection*): es una de las estrategias más utilizadas. Escoge de forma aleatoria tantos individuos de la población como el tamaño del torneo previamente definido (con o sin reemplazamiento). Los integrantes del grupo se enfrentan en "torneos" para finalmente obtener un ganador, el mejor individuo del grupo; cuantos más individuos en el torneo, menor probabilidad de seleccionar a los peores individuos. Normalmente se utiliza una versión probabilística en la cual se permite la selección de individuos que no sean los mejores de cada grupo.
- **Selección por ruleta** (*Roulette-wheel selection*): es el método más empleado para la selección de los progenitores. Da oportunidad a todos los individuos de ser seleccionados, ya que se asigna una probabilidad de selección proporcional al nivel de adaptación del individuo, y finalmente la selección se hace de forma aleatoria; teniendo nuevamente, los individuos mejor adaptados una mayor probabilidad de ser elegidos pero sin impedir la elección de los peor adaptados.
- **Ranking Lineal** (*Lineal Ranking*): parecido a los anteriores, este método ordena la población en función de su *fitness*, esto se asocia a una probabilidad de selección, dando un mayor peso a los mejores individuos sin dejar a un lado los peores, es decir, se tiene en cuenta la población en su totalidad.
- **Emparejamiento Variado Inverso** (*Negative Assortative Mating*): este método busca seleccionar progenitores cuyo material genético sea lo menos parecido posible. Para ello, se selecciona de forma aleatoria uno de los progenitores para posteriormente, de entre  $N$  individuos se escoge aquél que tenga un cromosoma más diferente.

**Cruce** Este operador entra en acción una vez se han seleccionado los progenitores; se escogen pares de "padres" que serán combinados siguiendo alguno de los métodos o técnicas de cruce, para finalmente generar dos soluciones conocidas como "hijos". Los descendientes o hijos contienen material genético de ambos padres, convirtiéndose en parte de la nueva y futura generación. Este operador equivale a la fase reproductiva de la evolución natural, y al partir de individuos ya existentes en la población, se puede intuir que es un operador de búsqueda local, es decir, que promueve la explotación; al favorecer el cruce entre los individuos mejor adaptados, se termina explorando las áreas más prometedoras del espacio de soluciones, pero sin descartar la posibilidad de obtener individuos con alta capacidad adaptativa al combinar o cruzar individuos no tan bien adaptados.

Este operador está asociado a una probabilidad, es decir, no todos los pares de progenitores tendrán descendencia, sino que esta elección es una decisión probabilística; normalmente, la probabilidad de este operador es alta, pero es importante controlarlo ya que con el paso de las generaciones, las poblaciones pueden ir perdiendo diversidad y caer en un óptimo local. Algunas de las técnicas más utilizadas para alfabetos finitos son:

- **Cruce en un punto**: a partir de dos progenitores, se dividen sus cadenas por un punto elegido al azar, lugar de cruce; dentro de las cadenas, existen  $l$  posibles bits o genes para ser el lugar de cruce. Finalmente de cada parente se obtienen dos subcadenas, y una de ellas se intercambiará con la otra subcadena del otro progenitor, de esta forma ambos descendientes heredan genes de cada uno de los padres, como se puede ver representado en la Fig. 3.30.

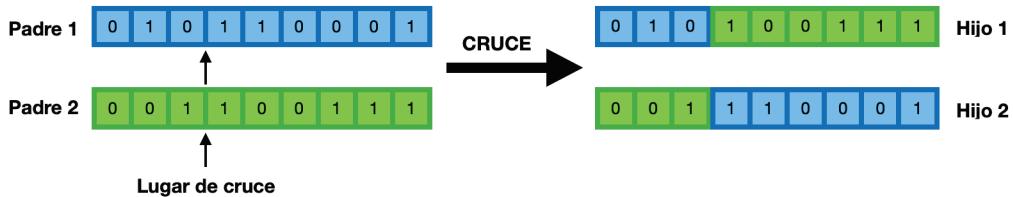
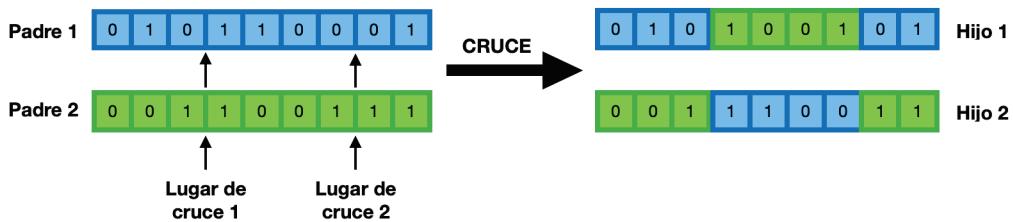


Figura 3.30: Ejemplo de cruce en un punto

- **Cruce en  $N$  puntos:** generalización de cruce en un punto; se seleccionan  $N$  puntos o lugares de cruce y se intercambia el material genético (Fig. 3.31).


 Figura 3.31: Ejemplo de cruce en  $N$  puntos, siendo  $N = 2$ 

- **Cruce uniforme:** este operador combina la información genética de forma homogénea. Para ello, se basa en la definición de una máscara de cruce expresada en forma de cadena de bits aleatoria de longitud  $l$ . Una vez más, se realiza una selección probabilística de cada gen, definiendo cual debe ser intercambiado y cual no (Fig. 3.32). El sesgo posicional es eliminado en este método, ya que la herencia de un gen es independiente de su posición dentro de la cadena.

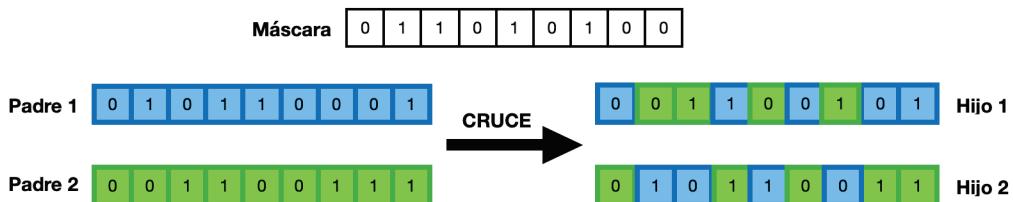


Figura 3.32: Ejemplo de cruce uniforme

**Mutación** Este operador también actúa bajo cierta probabilidad  $p_m$ , parámetro a definir por el desarrollador o diseñador del algoritmo y que dependerá del problema a resolver. Este operador se aplica de forma individual, es decir, la selección se hace individuo a individuo y se decidirá si aplicar el operador según la probabilidad  $p_m$ ; la mutación se aplica a un individuo progenitor y se obtiene un individuo descendiente. Otro factor a tener en cuenta es la probabilidad de mutación que se recomienda que sea relativamente baja, ya que la mutación es un proceso básico que introduce cierta aleatoriedad, enfocado en la exploración y búsqueda de la diversidad, por lo que si esta probabilidad es muy alta, el método se reduciría a una exploración aleatoria y perdería la esencia o carácter evolutivo; pero si se elige una probabilidad de mutación demasiado pequeña, habría una pérdida de diversidad genética cuando los individuos de la población estén convergiendo, pudiendo llevar al algoritmo a un subóptimo u óptimo local. La búsqueda del valor óptimo para la probabilidad de mutación, es una cuestión que ha sido motivo de muchos trabajos de investigación [218, 219], por ejemplo una de las recomendaciones es la utilización de una probabilidad  $p_m = 1/l$ , siendo  $l$  la longitud de la cadena [220]. Cuando a un individuo se le aplica la mutación, una de las metodologías más extendidas consiste en elegir aleatoriamente un lugar de mutación. A su vez, una vez elegido este lugar, se fija otra probabilidad que definirá la posibilidad de que ese gen mute o no, es decir, si se modifica el valor del gen por otro, elegido de forma aleatoria, que pertenezca al alfabeto;

en la codificación binaria, la mutación de un gen es tan simple como intercambiar los 0 por 1 y viceversa, como se aprecia en la Fig. 3.33.



Figura 3.33: Ejemplo de aplicación del operador de mutación

### 3.3.1.3. Reemplazo

El reemplazo es la fase del ciclo evolutivo donde se sustituyen todos los progenitores de la población, o parte de ellos, por sus descendientes, creando una nueva población para la siguiente generación. Existen diferentes planteamientos y modelos:

- **Modelo generacional:** se caracteriza porque en cada generación se crea una población totalmente nueva y completa que sustituye íntegramente a la anterior.
- **Modelo estacionario:** donde los nuevos individuos, es decir, los descendientes, reemplazan a todos o parte de los progenitores. Los modelos estacionarios buscan que los mejores individuos, los mejor adaptados, perduren tras varias generaciones, manteniendo y dando la posibilidad de utilizar su material genético en las siguientes poblaciones. Dentro de estos modelos existen diferentes estrategias de reemplazo, una de las más utilizadas es el reemplazo elitista o con elitismo (Subsec. 3.3.1.3).

**Elitismo** El elitismo consiste en conservar los mejores individuos a lo largo del proceso evolutivo, así este material genético se mantiene a lo largo de las generaciones y participa activamente en el proceso reproductivo que dará lugar a generaciones futuras; en definitiva, el aprendizaje elitista centra la búsqueda en las partes del espacio de búsqueda que previamente han destacado y se han descubierto como más prometedoras.

Como se ha explicado en 3.3.1.2, los operadores de selección tienen una base probabilística que favorece la conservación e incorporación de los mejores individuos de cada generación de cara a las futuras poblaciones, pero sin asegurar la supervivencia de estos individuos sobresalientes. Por lo que el elitismo es una estrategia que protege la preservación de los mejores individuos a lo largo del ciclo evolutivo y las sucesivas generaciones, ayudando a los algoritmos a converger en un óptimo global de forma más rápida y efectiva.

### 3.3.1.4. Evaluación

La función de evaluación, también llamada función objetivo o *fitness*, se diseña para cada problema concreto, no existe una función de evaluación general que se adecue a cualquier problema. La definición de esta función es uno de los puntos más importantes a la hora de diseñar un algoritmo evolutivo, ya que una mala definición de la *fitness* puede llevar a un mal funcionamiento del algoritmo, a problemas de optimización y una convergencia errónea. Esta función expresa el nivel de adaptación o adecuación de los individuos al problema; la *fitness* evalúa las soluciones, por lo que actúa sobre el individuo decodificado (fenotipo). Otro aspecto a tener en cuenta es que la función objetivo debe presentar cierta regularidad, es decir, que verifique que dos individuos cercanos en el espacio de búsqueda tengan también dos valores de *fitness* parecidas o cercanas.

En problemas de optimización muy restrictivos, existe un gran número de individuos o puntos en el espacio de búsqueda que representan soluciones no válidas cuyo valor o nivel de adaptación no es posible calcularlo, lo que se conoce como individuos no válidos. Los individuos no válidos suponen una gran dificultad a la hora de aplicar la computación evolutiva, por eso, existen diferentes técnicas o medidas correctivas como: la absolutista, consiste en aplicar operadores genéticos sobre los individuos no válidos hasta que estos sean válidos y se pueda

calcular su *fitness*; otra variante es la opción de asignar a los individuos no válidos una *fitness* muy mala para que sea el propio algoritmo el que los elimine durante el ciclo evolutivo; por ejemplo, otra opción sería aplicar un operador denominado reparador.

**CE para el diseño de redes neuronales** Como se ha comentado, la función de evaluación se diseña específicamente según el problema particular. En este apartado se presenta el caso particular de este proyecto, donde los individuos codifican soluciones del espacio de búsqueda, más concretamente, redes neuronales convolucionales (CNNs, Sec. 3.1.2.2). En este caso la función evaluación se puede entender como el proceso de entrenamiento de la propia red, es decir, del individuo una vez se ha decodificado, y el valor de adaptación o *fitness* se puede obtener a partir de diferentes métricas de rendimiento de las redes como por ejemplo la precisión o *accuracy*, la pérdida o *loss*, etc. El ciclo evolutivo se respeta y coincide con el seguido para cualquier otro problema, siendo como se muestra en la Fig. 3.34.

Para este caso concreto sigue presente el problema de los individuos no válidos (puntos fuera del espacio de búsqueda), por eso mismo la codificación utilizada es una elección de vital importancia como ya se ha comentado anteriormente (Sec. 4.3.1). Se busca tener una codificación consistente, sencilla y diversa, que facilite la representación del espacio de búsqueda al completo, que simplifique los cálculos computacionales, y que esté bien limitada. Con esto se busca evitar algunas situaciones irregulares que entorpecen el proceso de optimización, como los individuos no válidos, en este caso la generación de redes no válidas; o como el hecho de que individuos diferentes, es decir, con cromosomas diferentes, representen el mismo punto en el espacio de búsqueda, la misma red.

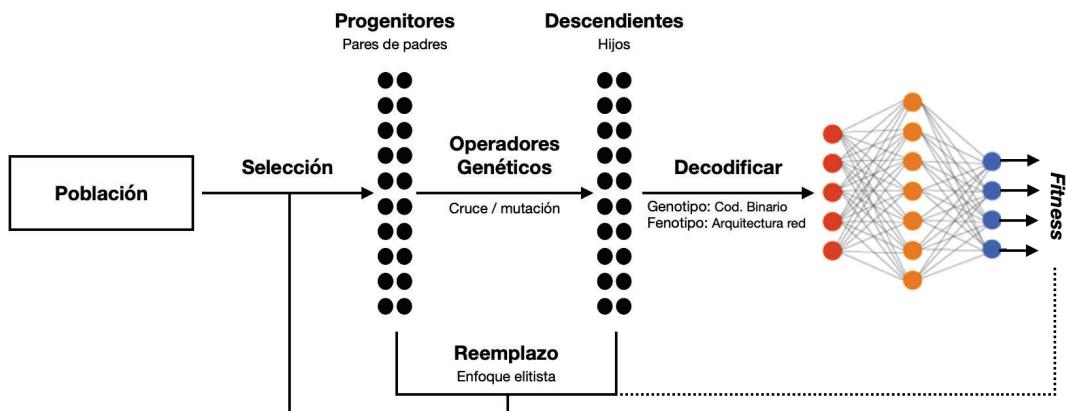


Figura 3.34: Ejemplo del flujo evolutivo de un problema de diseño de redes neuronales

### 3.3.2. Algoritmos evolutivos Multiobjetivo

Cuando un problema de optimización tiene varias funciones objetivo, es decir, tiene la tarea de encontrar una o más soluciones óptimas, se denominada optimización multiobjetivo o toma de decisiones multicriterio; se deben definir algunos criterios con el fin de determinar qué soluciones se consideran de buena calidad y cuáles no. El hecho de tener numerosos frentes a satisfacer, hace que existan soluciones *trade off* o escenarios de conflicto entre los diferentes objetivos; por lo que una solución óptima debe cubrir, en cierta medida, todos los objetivos, estableciendo así un compromiso entre los objetivos. Introduciendo así el concepto de dominancia; se dice que una solución  $x_1$  domina a otra  $x_2$ , siempre que se cumplan estas condiciones al completo:

1. La solución  $x_1$  es estrictamente mejor que  $x_2$ , por lo menos en un objetivo.
2. Cuando la solución  $x_1$  no sea peor que  $x_2$  en todos los objetivos.

Cuando se realiza un estudio de dominancia entre soluciones y no se puede demostrar alguna de las dos condi-

ciones anteriormente explicadas, no se puede concluir que una domine a la otra, se dice que las soluciones son no-dominadas. Por lo que el espacio de soluciones factibles está compuesto por puntos que satisfacen todos los objetivos y criterios marcados, y también contiene un conjunto de soluciones que son no-dominadas entre ellas, y este conjunto a su vez, tiene la propiedad de dominar al resto de soluciones del espacio que no pertenezcan al conjunto de soluciones no-dominadas, también denominado Frente de Pareto [1]; es decir, si se tiene un conjunto de soluciones  $P$  y un conjunto de soluciones no-dominadas  $P'$ , todos los puntos que pertenecen a  $P'$  no son dominados por ningún punto del conjunto  $P$ , siendo  $P'$  el frente de Pareto (Fig. 3.36).

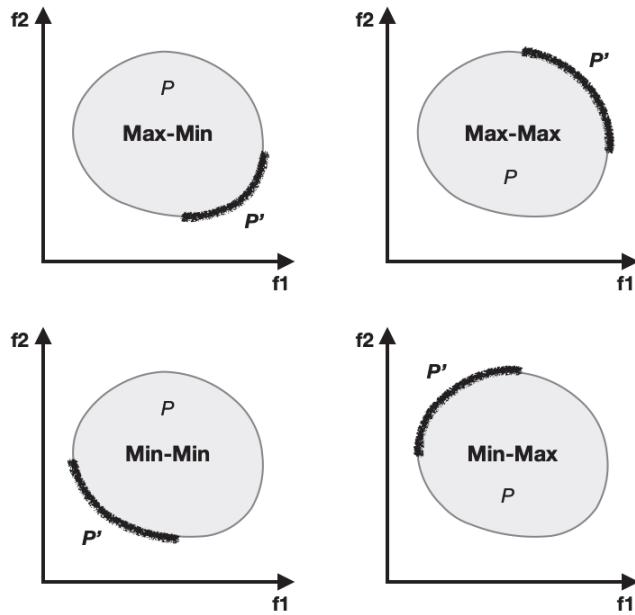


Figura 3.35: Ejemplo de diferentes frentes de Pareto para el mismo espacio de soluciones

Los problemas con múltiples objetivos se encuentran presentes en prácticamente en todas las disciplinas, representando un gran reto hasta en la actualidad [221, 222], ya que uno de sus objetivos es encontrar tantas soluciones Pareto-óptimas como sea posible. Uno de los métodos más novedosos y utilizados para la optimización de este tipo de problemas son los enfoques evolutivos, cuya solución se basa en la búsqueda de un punto óptimo a partir de una población; aunque esta tenga un tamaño relativamente pequeño con respecto al espacio de búsqueda, estos algoritmos son métodos útiles y eficaces, capaces de obtener un conjunto de soluciones finales de buena calidad.

### 3.3.2.1. NSGA-II

El NSGA-II, *Non Dominated Sorting Genetic Algorithm*, es un algoritmo evolutivo multiobjetivo elitista propuesto por K. Deb y sus estudiantes en el 2000 [223, 224]; es una versión mejorada del NSGA [225] que utiliza un operador de *crowding distance*, en vez de usar nichos. Este algoritmo se caracteriza por incluir el uso de mecanismos elitistas que preservan la diversidad y su foco en las soluciones no dominadas.

El NSGA-II ejecuta la fase de selección entre la población de padres junto con el conjunto de hijos o descendientes; partiendo de una población  $N$ , el proceso de selección se realiza sobre una nueva población de tamaño  $2N$ . Sobre esta población conjunta, se ordenan los individuos no-dominados y se clasifica la población en diferentes frentes de Pareto que se encuentran en constante modificación a lo largo del proceso evolutivo; aunque esto requiera de un mayor esfuerzo, así se permite realizar una verificación global de dominancia entre la población de padres y descendientes. La nueva población formada a partir de los individuos seleccionados, se empieza a construir a partir de los frentes no dominados, comenzando por el mejor ( $F_1$ ), continuando con el segundo ( $F_2$ ) y así sucesivamente, según la tendencia elitista característica de este algoritmo; esta nueva población man-

tendrá el tamaño inicial  $N$ , por lo que no se llegarán a tener todos los frentes en cuenta, haciendo que estos frentes desaparezcan, como se aprecia en la figura 3.36.

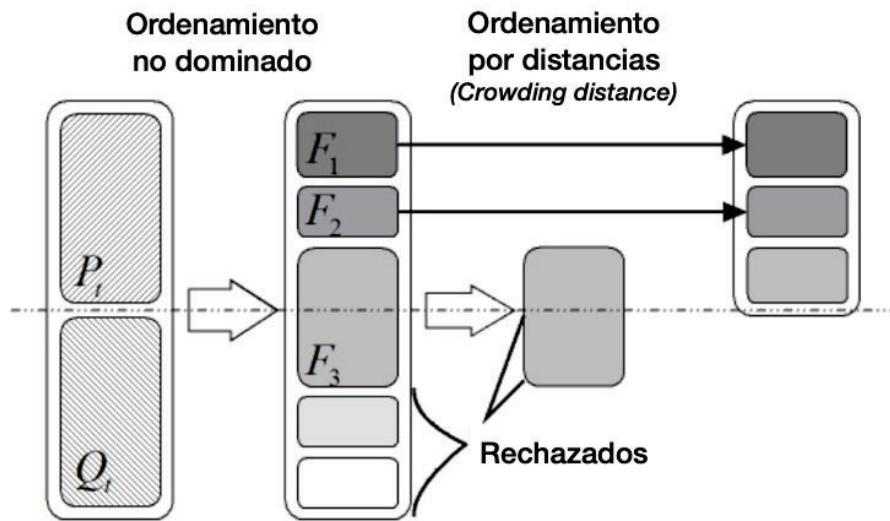


Figura 3.36: Esquema del mecanismo de promoción/preservación de individuos del NSGA-II

## Capítulo 4

# Diseño y Metodología

### 4.1. Herramientas

En esta sección se detallan las herramientas y materiales utilizados para la realización y desarrollo del proyecto. Está dividida en cuatro apartados: entorno de ejecución (Subsec. 4.1.1), hace referencia al servidor externo donde se han enviado y ejecutado los experimentos; hardware (Subsec. 4.1.1.2), es el conjunto de elementos físicos o materiales utilizados; software (Subsec. 4.1.1.3), es el conjunto de programas y versiones necesarias para el desarrollo de los experimentos; y el conjunto de datos o *dataset* con los que se ha trabajado (Subsec. 4.1.2).

#### 4.1.1. Entorno de ejecución

Debido a los altos requerimientos computacionales que implican los experimentos, se hizo uso de los servidores externos de alta computación CESGA [13].

##### 4.1.1.1. CESGA

La Fundación Pública Galega Centro Tecnológico de Supercomputación de Galicia (CESGA) es el centro de cálculo, comunicaciones de altas prestaciones y servicios avanzados de la Comunidad Científica Gallega, Sistema Académico Universitario y del Consejo Superior de Investigaciones Científicas (CSIC). La Fundación tiene como misión contribuir al avance de la Ciencia y la Tecnología; promueven la investigación y el uso del cálculo intensivo, comunicaciones avanzadas y desarrollo de las tecnologías de la información y comunicaciones, como instrumento para el desarrollo socioeconómico sostenible.

FinisTerrae es el nombre genérico de las distintas generaciones de superordenadores del Centro de Supercomputación de Galicia; en el año 2007 se instaló el primer equipo de la serie FinisTerrae que alcanzó la posición número 100 en la lista de los 500 ordenadores más potentes del mundo. El equipo actualmente en servicio y utilizado para este proyecto, el FinisTerrae III [12] fue instalado en el año 2021 y puesto en producción en el año 2022. Es un equipo Bull ATOS bullx distribuido en 13 racks o armarios, con una potencia de cómputo total de 4 Peta-FLOPS y formado por 354 nodos interconectados a través de una red Infiniband HDR con 708 procesadores Intel Xeon Ice Lake 8352Y de última generación con 32 cores a 2,2Ghz (22.656 cores) y 144 GPUs (128 Nvidia A100 y 16 Nvidia T4), como se puede ver representado en la figura 4.1 (Fig. 4.1).

Los experimentos se han realizado en nodos GPU (*GPU nodes*), las características de estos nodos se explican en más detalle en la subsección 4.1.1.2. El entorno se caracteriza por trabajar con un sistema de colas [226] destinado a realizar simulaciones con múltiples cores, grandes cantidades de memoria y tiempo; gracias a este sistema, se pueden reservar y garantizar los recursos asignados para cada usuario. Este sistema de colas está basado en SLURM [227], que como gestor de cargas de trabajo de clúster tiene tres funciones clave: asignar acceso exclusivo o no exclusivo de los recursos a los usuarios durante un determinado tiempo; proporcionar un marco

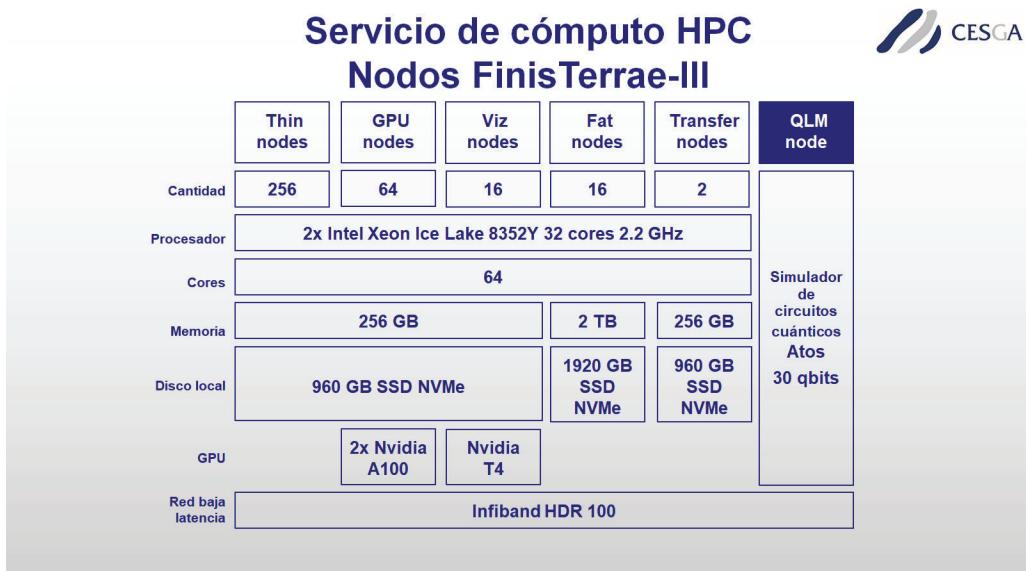


Figura 4.1: Distribución del supercomputador FinisTerrae III [12]. Imagen de [13]

para iniciar, ejecutar y supervisar los diferentes nodos y sus trabajos; y administrar la cola de trabajos pendientes y los recursos disponibles. En el caso particular del desarrollo de este proyecto, la disponibilidad y uso de los recursos tenía una duración máxima de 168 horas, es decir, siete días. Para la conexión y acceso al servidor cesga, es necesario darse de alta como usuario; la conexión y manejo de los servicios se ha realizado en su totalidad desde la terminal o *shell* de macOS con un cliente SSH (*Secure Shell*) que permite transferencias de información encriptada [228, 229].

Como se ha comentado anteriormente, el servidor está basado en SLURM que proporciona un marco de trabajo y manejo de los trabajos (*jobs*). Para poder enviar y poner en marcha un experimento, es necesario utilizar algunos de los comandos básicos de SLURM que podemos encontrar explicados en [230]. A continuación, se adjunta un ejemplo de cómo se envían los trabajos a los nodos a partir de un script de bash (*.sh*):

```

1  #!/bin/bash
2  #SBATCH -n 1
3  #SBATCH -t 168:00:00
4  #SBATCH -c 64
5  #SBATCH --gres=gpu:2
6  #SBATCH -p cola-corta,thin-shared,thinnodes,gpu-shared,gpu-shared-v100
7  #SBATCH --mem=200G
8  #SBATCH --mail-type BEGIN, END, FAIL
9  #SBATCH --mail-user=m.apellaniz.portos@gmail.com
10
11 module load cesga/2020 gcccore/system
12 module load python/3.6.12
13
14 jupyter nbconvert --to notebook --ExecutePreprocessor.timeout=600000 --output-dir="./
    resultados" --execute GA_notebook.ipynb

```

Basándonos en el código 4.1.1, para poder lanzar los experimentos es necesario definir los recursos necesarios para el correcto funcionamiento del sistema de colas:

- **-n:** número de tareas (*tasks*)
- **-t:** tiempo de uso de los recursos (formato HH:MM:SS)
- **-c:** número de núcleos (*cores*)

- **--gres=gpu**: petición de uso de GPUs. Se puede definir el número de GPUs deseadas, p.ej. gres=gpu:2.
- **-p**: solicitud de una partición específica. Se pueden solicitar varias separadas por comas.
- **--mem**: (o *--mem-per-cpu*) parámetro obligatorio, determina la memoria demandada para el trabajo.
- **--mail-type**: parámetro opcional, define el tipo de notificaciones que te interesan. En este caso, saber cuándo comienza, cuándo termina, o si ha fallado.
- **--mail-user**: correo electrónico en el que se desea recibir las notificaciones.

Para el desarrollo de diferentes trabajos, proyectos y aplicaciones, es necesario definir un entorno determinado que sea independiente de la *shell* del usuario; para ello existe el comando `module` (Fig. 4.2). Las aplicaciones disponibles son jerárquicas y están vinculadas a la precarga de una combinación concreta de dependencias. Esto busca minimizar los problemas de incompatibilidades entre las distintas librerías y los compiladores.

```
[csgpamap@login211-1 ~]$ module avail
-----[Core]-----
acrfinder/20200507          (Bio)    flatbuffers-python/2.0          (Tool)   mult卿c/1.10.1          (Bio)
admb/12.3                   (Comp)   flatbuffers/2.0.0           (Tool)   mummer/4.0.0rc1         (Bio)
admb/20210912                (Comp,D) flint/2.8.4              (Math)   muscle/3.8.1551         (Bio)
admixture/1.3.0              (Bio)    flye/2.8.3              (Bio)    myqlm/1.1.6-python-3.6.12  (QComp)
advisor/2021.4.0              (Prof)   flye/2.9                (Bio,D)  myqlm/1.2.2-python-3.6.12  (QComp)
```

Figura 4.2: Captura de pantalla al ejecutar el comando para revisar los módulos disponibles en FinisTerrae III.

### 4.1.1.2. Hardware

A lo largo del proyecto se han desarrollado diferentes experimentos con distinto hardware; experimentos sin GPU y con una o varias GPUs.

Especificaciones	Tipo
CPU	Procesador Intel® Xeon® Platinum 8352Y
Número de núcleos	32
Número de subprocesos	64
Frecuencia turbo máxima	3.40 GHz
Frecuencia básica del procesador	2.20 GHz
Caché	48 MB
Intel® UPI Speed	11.2 GT/s
Potencia de diseño térmico (TDP)	205 W

Tabla 4.1: Especificaciones del software utilizado en el proyecto

Especificaciones	Tipo
GPU	a100 NVIDIA A100-PCIE-40GB
Versión del driver de CUDA / Versión Runtime	11.5 / 11.2
Capacidad CUDA - versión	8.0
Memoria	40 GB of HBM2 BANDWIDTH 1555 GB/s
Multiprocesadores	108
Núcleos CUDA	64 CUDA Cores/MP - 6912 CUDA núcleos total
Frecuencia máxima del reloj GPU	1410 MHz (1.41 GHz)

Tabla 4.2: Especificaciones del hardware utilizado en el proyecto

#### 4.1.1.3. Software

- **OS**: el principal sistema operativo utilizado para el desarrollo práctico de este proyecto ha sido Linux [231]; es uno de los sistemas operativos más populares en el mundo de la computación, de tipo Unix compuesto por software libre y de código abierto. Sobre todo se ha hecho uso de la *Shell* de Linux para enviar, revisar, acceder y editar los *scripts* y experimentos ejecutados.

Para el proceso teórico, de investigación, lectura, escritura y pequeños experimentos de ajuste de código, se ha utilizado en local, con un sistema operativo local de macOS Monterey 12.2.1 [232].

- **Overleaf** [233]: es una herramienta de publicación, edición y redacción de texto colaborativa en línea, enfocado a documentos científicos para facilitar su elaboración. Overleaf es un editor basado en LaTeX, fácil de usar de forma colaborativa en tiempo real y con una compilación automática en segundo plano.
- **Anaconda** [234]: es una distribución de los lenguajes de programación Python y R para computación científica, que tiene como objetivo simplificar la administración y el despliegue de paquetes. La distribución incluye paquetes de ciencia de datos adecuados para Windows, Linux y macOS. Es desarrollado y mantenido por Anaconda, Inc., y fue fundada por Peter Wang y Travis Oliphant en 2012. Las versiones de paquetes en Anaconda son administradas por el sistema de administración de paquetes conda, desarrollado como un paquete de código abierto separado. También hay una pequeña versión de arranque de Anaconda llamada Miniconda [235].
- **Python** [236]: es un lenguaje de programación de alto nivel, interpretado, multiplataforma y de propósito general cuya filosofía enfatiza la legibilidad del código. Es multiparadigma, ya que está orientado a objetos y soporta programación imperativa y funcional. Es administrado por *Python Software Foundation* y tiene una licencia de código abierto. Este lenguaje es uno de los lenguajes de programación más populares en el campo científico y de investigación, principalmente por su completa biblioteca y la facilidad de creación de prototipos; de hecho, es el lenguaje más popular en los campos de AI y DL.
- **Tensorflow** [237]: es una biblioteca de software gratuita y de código abierto enfocada al aprendizaje automático. Dispone de un conjunto completo de herramientas, bibliotecas y recursos que permiten implementar, probar y desarrollar fácilmente aplicaciones de ML. Ha sido desarrollado por el equipo de *Google Brain*, que incluyó esta biblioteca dentro del desarrollo de la mayoría de los productos y proyectos de investigación de Google. Con el lanzamiento de la versión 2.0.0 en 2019, TensorFlow se ha vuelto más simple y fácil de usar.
- **Keras** [238]: es una biblioteca de código abierto escrita en Python y capaz de ejecutarse sobre otras bibliotecas de ML como TensorFlow, Microsoft Cognitive Toolkit o PlaidML. TensorFlow lo implementa en su biblioteca central desde 2017 para facilitar aún más el diseño, desarrollo e implementación de aplicaciones al combinar las capacidades gráficas sencillas de Keras y la potencia y el rendimiento de TensorFlow.
- **CUDA** [239]: plataforma informática paralela que incluye un compilador y un conjunto de herramientas de desarrollo creadas por Nvidia, que permite a los desarrolladores utilizar una GPU compatible con CUDA para realizar operaciones matemáticas computacionalmente intensas, mucho más rápido de lo que lo haría una CPU.
- **DEAP** [240]: desarrollado en el Laboratorio de Sistemas y Visión por Computador (CVSL) de la Universidad de Laval, Quebec, Canadá. DEAP es un framework evolutivo destinado al desarrollo de prototipos de forma rápida y sencilla; busca que los algoritmos sean explícitos y las estructuras de datos transparentes. Adaptable a mecanismos de paralelización como *multiprocessing* y SCOOP.

#### 4.1.2. Conjunto de datos

El conjunto de datos utilizado para este proyecto es un dataset presentado en [14] por Olsen et al.; este conjunto de datos también se encuentra disponible dentro de la colección de datasets de Tensorflow [241], para su correcto funcionamiento y carga es imprescindible instalar y utilizar el paquete “*tfds-nightly*” [242] que se actualiza diariamente e incluye las últimas versiones de los conjuntos.

El dataset está compuesto por 17,509 imágenes etiquetadas de ocho especies diferentes de malas hierbas de gran importancia nacional y nativas de ocho lugares del norte de Australia (Fig. 4.4). En [14], entrena un clasificador de imágenes de aprendizaje profundo para conseguir identificar las especies presentes en cada imagen y así validar en tiempo real el rendimiento del clasificador, intentando replicar el supuesto e hipotético comportamiento de un robot autónomo especializado en el control de malezas. Estas especies fueron seleccionadas debido a su idoneidad para la pulverización de herbicidas foliares y su notoriedad; estas destacan por su alto nivel de invasividad y propagación, e impacto socioeconómico dañino de las zonas rurales de Australia.

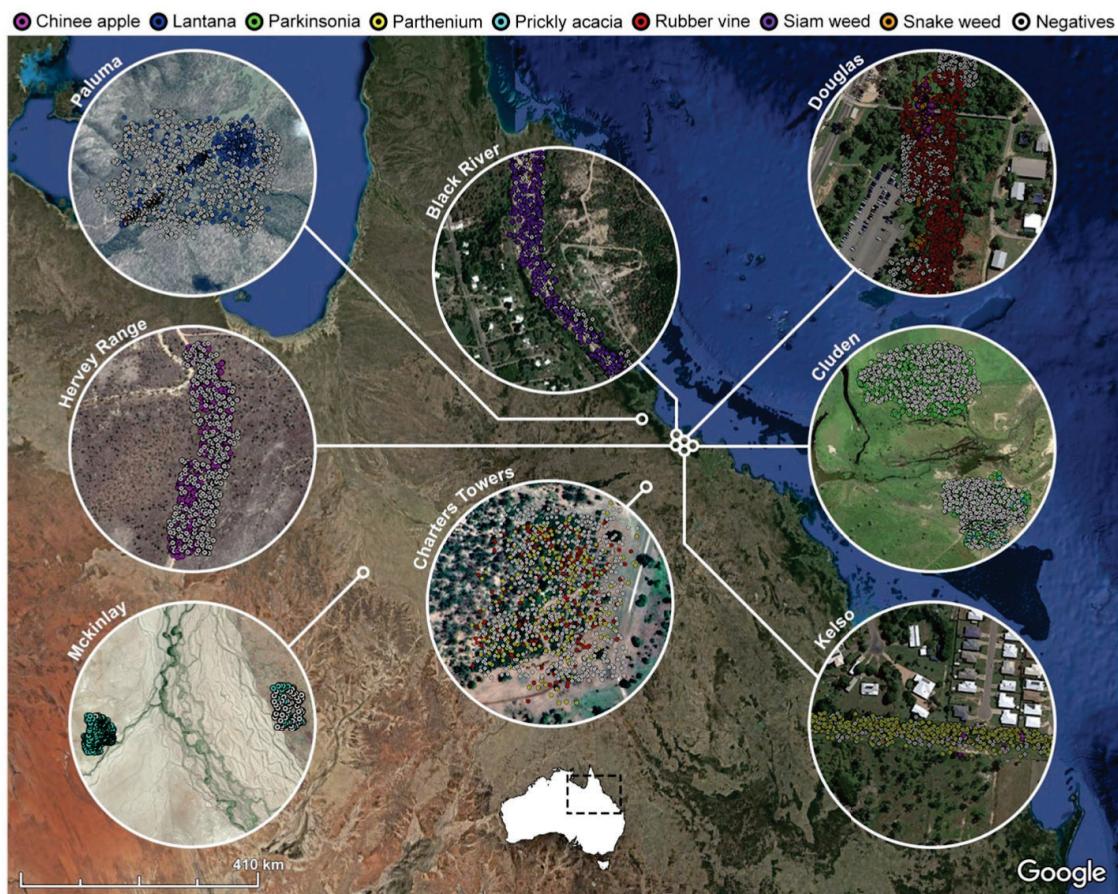


Figura 4.3: Distribución geográfica de las imágenes de malas hierbas en el norte de Australia (Datos: Google, SIO, NOAA, U.S. Navy, NGA, GEBCO; Imagen Landsat/Copernicus 2018; Image DigitalGlobe 2018; Imagen CNES/Airbus 2018). Imagen de [14]

El problema de clasificación de maleza o malas hierbas, sobre todo en ambientes pastizales, no es un problema trivial; los entornos de pastizales plantean desafíos únicos para el manejo y la clasificación de malezas porque son remotos y extensos, terrenos accidentados y desiguales. Además, existe la principal dificultad debida a la aleatoriedad y diversidad del ambiente, ya que coexisten diferentes especies de malas hierbas y plantas nativas en un mismo área; también hay otros aspectos técnicos a tener en cuenta, como la distancia del objeto a la cámara, los niveles de luminosidad, sombras o incluso objetos parcial o completamente ocultos.

Olsen et al. establecieron diversos objetivos para conseguir la variabilidad, diversidad y generalidad del conjunto de datos. Para ello, se han recopilado al menos 1000 imágenes de cada especie; también, para conseguir una división equitativa, 50-50, entre una clasificación positiva o negativa, es decir, entre imágenes con alguna de las ocho especies objetivo o imágenes sin la presencia de estas hierbas, se ha incluido una novena clase. Esta clase "negative" es la clase mayoritaria, compuesta por 9106 imágenes y que representa al resto de plantas, es decir, aquellas que no pertenecen a las ocho clases seleccionadas. Esta estrategia se toma para evitar el ajuste excesivo de los modelos a las características de la imagen a nivel de escena. A continuación se muestra una tabla resumen con las ocho especies de maleza que componen el conjunto de datos, tabla 4.1.2; y una gráfica con la distribución del dataset, (Fig. 4.4).

Especie	Clase	Nº de ejemplos	Porcentaje
Manzana china ( <i>Ziziphus mauritiana</i> )	0	1125	6.4 %
Lantana ( <i>lantana camara</i> )	1	1064	6.4 %
Parkinsonia ( <i>parkinsonia aculeata</i> )	2	1031	5.9 %
Partenio ( <i>Partenio hysteroforo</i> )	3	1022	5.8 %
Acacia espinosa ( <i>Vachellia nilotica</i> )	4	1062	6.1 %
Enredadera de caucho ( <i>Criptostegia grandiflora</i> )	5	1009	5.8 %
Hierba siam ( <i>Chromolaena odorata</i> )	6	1074	6.1 %
Hierba de serpiente ( <i>Stachytarpheta spp.</i> )	7	1016	5.8 %
<i>Negative</i>	8	9106	52.0 %

Tabla 4.3: Distribución de clases dentro del conjunto de datos

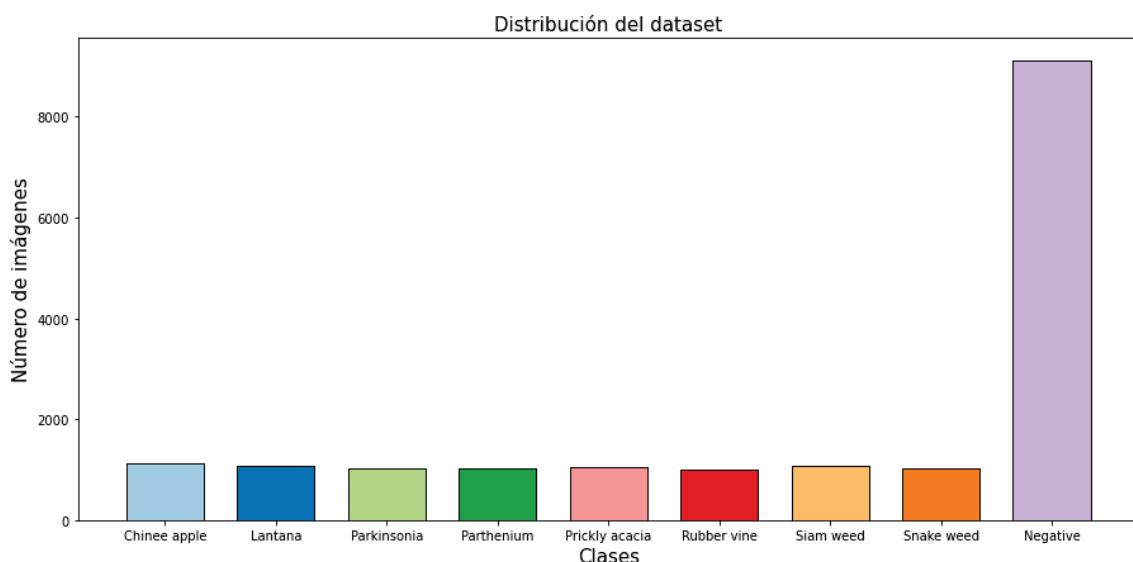


Figura 4.4: Gráfico con la distribución de las clases en el dataset.

## 4.2. Preprocesamiento de los datos

Para entrenar modelos con imágenes es necesario someterlas a un procedimiento de tratamiento para poder trabajar con la información que tienen de una forma más ágil y sencilla.

En este proyecto se trabaja con imágenes .jpg de 256x256 RGB, con tres canales; para agilizar los cálculos y coste computacional, se opera con matrices. Como en [14], se decide realizar una división del dataset original de 60-20-20 % para entrenamiento, validación y test, respectivamente. Las imágenes de entrenamiento y validación

serán las únicas que estarán en contacto directo con el modelo durante el proceso de entrenamiento, siendo el objetivo final del subconjunto de validación el monitoreo del entrenamiento y minimización del sobreajuste, es el encargado de la actualización de los pesos tras cada iteración o *epoch*; en cambio, el conjunto de test se utilizará a posteriori, únicamente con los modelos ya entrenados. Se ha decidido realizar una división igual a la aplicada por Olsen et al. para que la futura comparación de resultados sea lo más coherente posible, ya que estos se habrán obtenido o partirán de entrenamientos sometidos a una situación y condiciones lo más similares posible. Para esta división se ha utilizado la librería de Tensorflow “*train\_test\_split*” [243], se realizó una partición aleatoria estratificada (“*stratify*”) para garantizar una distribución uniforme de las clases dentro de cada subconjunto, excepto para la clase negativa que es la más numerosa, como se aprecia en la figura 4.5; estas divisiones aleatorias se controlan mediante una semilla aleatoria (“*random\_state*”), de modo que la división sea reproducible.

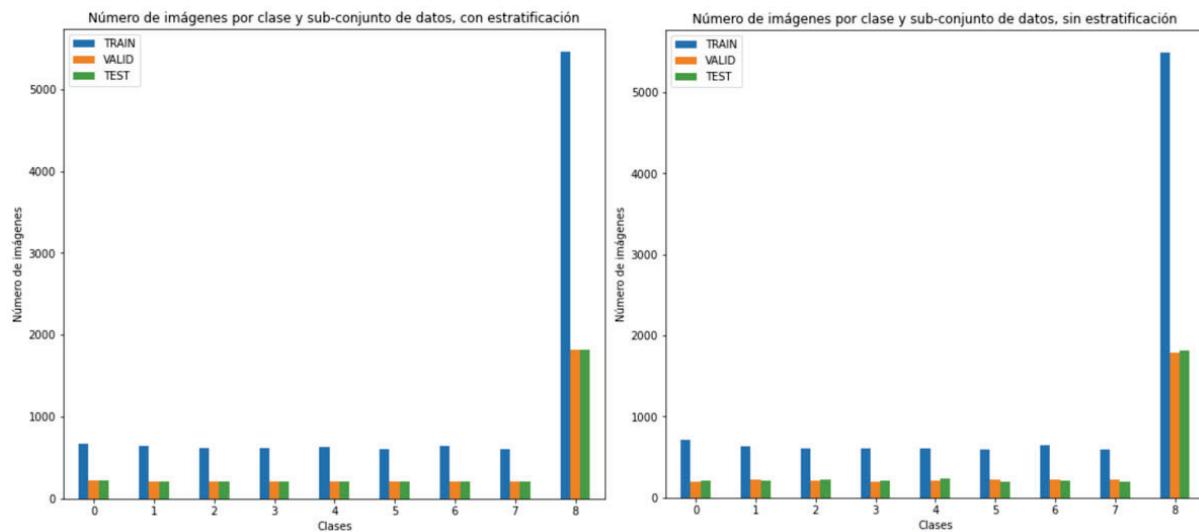


Figura 4.5: Gráfico con la distribución de las clases para cada subconjunto de datos tras la división del dataset original. A la izquierda el reparto resultante al seguir una división aleatoria estratificada, mientras que a la derecha se realiza una división meramente aleatoria simple.

En cuanto a la lectura y carga de las imágenes, se realiza con la librería PIL de Python [244], en concreto el módulo *Image*. Se normalizan los valores de la imagen, es decir, se trasladan a una escala entre 0 y 1 para facilitar las operaciones y su manejo; como se ha mencionado anteriormente, las imágenes son RGB por lo que para normalizarlas se debe dividir cada píxel de la imagen entre el valor máximo que este puede alcanzar, en este caso hasta 255 (rango RGB [0, 255]). Se realiza un *cast* para trabajar con *floats* y se reduce el tamaño de las imágenes a 224x224 píxeles, tamaño utilizado en las redes ResNet-50 e Inception-v3 3.1.2.2; esta reducción de tamaño no solo hace que nuestro dataset sea compatible con los modelos presentados en [14], sino que también reduce el número de píxeles de la imagen lo que se traduce en una leve reducción de la dificultad computacional.

```

1 def load_img(filename): #Funcion para abrir, leer y guardar como un array (matriz) las
2     imagenes
3     image_path= os.path.join(IMGS_DIR, filename)
4     image = Image.open(image_path)
5     image=np.array(image)
6     return image
7
8 def normalize_data(image): #Funcion para normalizar las imagenes. Recorte + cast
9     image = tensorflow.image.resize(image, IMAGE_SHAPE)
10    return tensorflow.cast(image, tensorflow.float32) / 255.

```

Como se ha explicado en 3.2.1.1, se utiliza la técnica de *Data Augmentation*; únicamente se aplica a los subconjuntos de entrenamiento y validación. En este caso se utilizan métodos como el giro a la izquierda y a la derecha,

el volteo arriba y abajo, y la rotación de 90°. No se aplica *data augmentation* en el subconjunto de test ya que estas imágenes se utilizan para evaluar el comportamiento de los modelos ya entrenados, por lo que es interesante ver cómo se comportan con imágenes reales, ver los resultados obtenidos al aplicar estos modelos a situaciones del mundo real.

```

1 def preprocess_train_val(image, labels): #Preprocesamiento de las imagenes de
2     entrenamiento y validacion (Normalizacion + Data Augmentation)
3     image = tensorflow.image.random_flip_left_right(image)
4     image = tensorflow.image.random_flip_up_down(image)
5     image = tensorflow.image.rot90(image)
6     image = normalize_data(image)
7     return (image, labels)
8
9 def preprocess_test(image, labels): #Preprocesamiento de las imagenes de test (
10    Normalizacion)
11     image = normalize_data(image)
12     return (image, labels)

```

Para la creación de los subconjuntos de datos y que estos sean compatibles con el entrenamiento distribuido, se ha utilizado el método *tensorflow.data.Dataset.from\_tensor\_slices* [245]. A la hora de crear los datasets, se mezclan de forma aleatoria las imágenes (*shuffle*); también se forman lotes con un tamaño determinado definido en el *.json* con los parámetros del algoritmo. El tamaño de los lotes dependerá del número de GPUs que se vayan a utilizar para el entrenamiento; por ejemplo, si el hardware del que disponemos tiene ciertas limitaciones y para ejecutar de forma correcta los experimentos solo admite hasta un *batch size* de 32 (un paquete o lote de 32 imágenes), si disponemos de 4 GPUs con las mismas condiciones, se formarán lotes de  $32 \times 4 = 128$  (32 imágenes para cada GPU). Por eso mismo, el tamaño de los *batches* variará en función del hardware disponible; para saber el número de recursos disponibles se utiliza *strategy.num\_replicas\_in\_sync*. Como se ha comentado en 4.1.1.1, CESGA tiene un límite de 2 GPUs por nodo, por lo que el tamaño de *batch* elegido y definido en *params.json* será el número de imágenes que procesará cada GPU, y a la hora de generar el dataset este tamaño se duplicará.

```

1 training_dataset = tensorflow.data.Dataset.from_tensor_slices((x_train_imgs, y_train))
2 val_dataset = tensorflow.data.Dataset.from_tensor_slices((x_val_imgs, y_val))
3 test_dataset = tensorflow.data.Dataset.from_tensor_slices((x_test_imgs, y_test))
4
5
6 training_dataset = (training_dataset
7     .shuffle(BUFFER_SIZE_TRAIN)
8     .batch(GLOBAL_BATCH_SIZE)
9     .map(preprocess_train_val, num_parallel_calls=tensorflow.data.experimental.
10          AUTOTUNE)
11     .prefetch(tensorflow.data.experimental.AUTOTUNE)
12 )
13
14 val_dataset = (val_dataset
15     .shuffle(BUFFER_SIZE_VAL)
16     .batch(GLOBAL_BATCH_SIZE)
17     .map(preprocess_train_val, num_parallel_calls=tensorflow.data.experimental.
18          AUTOTUNE)
19     .prefetch(tensorflow.data.experimental.AUTOTUNE)
20 )
21
22 test_dataset = (test_dataset
23     .shuffle(BUFFER_SIZE_VAL)
24     .batch(len(y_val))
25     .map(preprocess_test, num_parallel_calls=tensorflow.data.experimental.AUTOTUNE)
26     .prefetch(tensorflow.data.experimental.AUTOTUNE)
27 )

```

## 4.3. Algoritmo genético

En esta sección se describe la implementación y desarrollo del algoritmo genético, así como todos los aspectos importantes y necesarios para su correcto funcionamiento.

Para el desarrollo del código se ha utilizado la librería elaborada en Python, DEAP (*Distributed Evolutionary Algorithms in Python*) [246]. DEAP es una de las librerías más utilizadas para el desarrollo de algoritmos evolutivos gracias a su gran personalización, al elevado número de representaciones aceptadas y los diversos y numerosos algoritmos implementados dentro de la librería. Se ha seleccionado esta librería mayoritariamente por su filosofía, ya que a diferencia de otras librerías que utilizan inicializadores cerrados, en DEAP es el usuario quien debe definir y determinar cada uno de los parámetros del algoritmo, además, estos son fácilmente modificables e integrables con el resto del software, adaptándose correctamente a cada problema.

Esta sección se divide en tres apartados: la codificación de un individuo, donde se explica en detalle cómo se codifica el problema (Subsec. 4.3.1); a continuación se explica cómo se hace el cálculo de la fitness, es decir, la evaluación de los individuos (Subsec. 4.3.2); finalmente, se dedica un capítulo a otros aspectos importantes como son los diferentes operadores genéticos (Subsec. 3.3.1.2).

### 4.3.1. Codificación de un individuo

La codificación del problema es uno de los puntos más delicados del desarrollo de algoritmos genéticos. En este caso, se ha planteado una codificación binaria basada en la selección de diferentes parámetros y configuraciones para cada una de las capas que conforman la arquitectura de la red neuronal convolucional.

En este caso, cada individuo representa una arquitectura, por lo que su codificación trata de definir y seleccionar diferentes parámetros y configuraciones para las capa que conforman cada red neuronal convolucional. Un individuo está codificado por una serie de lotes de 10 bits binarios que representan cada capa de la arquitectura, llegando a alcanzar un cromosoma de longitud máxima de  $10 * n\_capas$  bits. Estos lotes dan información sobre el tipo de capa que es y algunos de sus parámetros más importantes; se pueden distinguir cuatro tipos de capas gracias a los dos primeros genes que definen el tipo de capa de la que se trata, como se puede ver en la Tabla 4.3.1

Tipo de capa	Posición de los genes (10 bits)									
	0	1	2	3	4	5	6	7	8	9
<i>Ghost</i>	o	o	-	-	-	-	-	-	-	-
<i>Pooling</i>	o	1	<i>T</i>	<i>K</i>	-	-	-	-	-	<i>S</i>
<i>Convolucional</i>	1	o	$K_{x1}$	$K_{x2}$	$K_{y1}$	$K_{y2}$	$FMS_1$	$FMS_2$	$A_T$	<i>S</i>
<i>Convolucional residual</i>	1	1	$K_{x1}$	$K_{x2}$	$K_{y1}$	$K_{y2}$	$FMS_1$	$FMS_2$	$A_T$	<i>S</i>

Tabla 4.4: Codificación de una capa de la CNN con el algoritmo genético

Esta codificación permite que se generen arquitecturas ya existentes como las comentadas en 3.1.2.2, y otras completamente nuevas; esto hace que estas arquitecturas clásicas también puedan participar en el ciclo evolutivo, pudiendo llegar a aparecer hasta como la mejor solución posible del espacio de búsqueda.

- **Capa fantasma (ghost layer):** son las capas más simples. Existen únicas y específicamente para poder codificar arquitecturas con diferente profundidad, es decir, con más o menos capas; estas capas no tienen ningún efecto directo en los entrenamientos, no realizan procesos ni cálculos.
- **Capa Pooling:** ya explicadas en Sec. 3.1.2.2, para su correcta configuración se definen tres parámetros principales: *T*, *K* y *S*.
  - **Tipo de pooling (T):** hace referencia al tipo de capa *pooling*, es decir, el tipo de operación que realiza. Puede ser: *Average Pooling* o *Max Pooling*.

- **Tamaño del kernel ( $K$ )**: este parámetro define el tamaño del filtro (kernel) para la operación, es decir, el número de píxeles, o celdas ya que trabajamos con matrices, que entran a la operación y que se reducirán a un único valor resultante. El kernel define una ventana cuadrada con un tamaño de 2 o 3 píxeles.
- **Stride o paso ( $S$ )**: define el paso con el que se recorre el kernel a lo largo de la imagen de entrada, este concepto se explica en más profundidad en Sec. 3.1.2.2. El paso puede tomar dos valores diferentes: stride de 2 o 3 píxeles.

Como se puede observar, siguiendo esta codificación pueden llegar a generar varias capas de *pooling* de forma consecutiva, algo que no tiene ningún sentido; por eso mismo, se ha limitado la aparición de este tipo de capas de forma consecutiva, de modo que si se codifican varias capas *pooling* seguidas solo se tendrá en cuenta la primera, ignorando las siguientes.

- **Capa Convolucional**: estas capas se han presentado en el marco teórico (Sec. 3.1.2.2). En comparación con las anteriores, este tipo de capas presentan más parámetros ajustables a tener en cuenta durante su definición, lo que da lugar a un mayor número de combinaciones posibles. En este tipo de capas el tamaño del kernel puede dar lugar a un filtro asimétrico, en este caso, se dará la posibilidad de que esta asimetría del kernel sea posible de codificar ( $K_x$  y  $K_y$ ).
  - **Tamaño del kernel en el eje de abscisas (horizontal) ( $K_x$ )**: consta de dos bits para su codificación, pudiendo generar hasta cuatro combinaciones de tamaños diferentes.
  - **Tamaño del kernel en el eje de ordenadas (vertical) ( $K_y$ )**: al igual que el  $K_x$ , este define el tamaño del kernel en la dirección del eje de ordenadas o eje y (vertical). Finalmente, el tamaño del filtro puede ser 1, 3, 5 o 7, y viene dado por la Ec. 4.3.1; así se demuestra que es posible que  $K_x$  y  $K_y$  sean diferentes, es decir, el filtro no sea cuadrado:

$$\text{Kernel\_size} = \text{dec}(K) * 2 + 1 \quad (4.1)$$

- **Número de filtros (Feature Map Size) (FMS)**: se refiere al número de filtros que existen en cada una de las capas de convolución, como se ha explicado anteriormente en Sec. 3.1.2.2. Un valor más alto de FMS teóricamente obtendrá una cantidad de detalles de la imagen mayor, como consecuencia, la red será más pesada y lenta, lo que implica que consumirá y necesitará una mayor capacidad computacional; esta dificultad se agrava en las capas iniciales, cuando la dimensión de las imágenes es muy grande. Para determinar el número de filtros que tendrá una capa, se debe tener en cuenta que la codificación de este proyecto es binaria (Ec. 4.3.1); los valores posibles que se pueden obtener son 64, 128, 256 y 512.

$$FMS = 2^p \quad (4.2)$$

$$p = \text{dec}(FMS) + 6 \quad (4.3)$$

- **Tipo de función de activación ( $A_T$ )**: como se menciona y explica en la Sec. 3.1.2.1, existen diferentes funciones de activación. El noveno gen de las capas convolucionales hace referencia al tipo de función de activación que tiene, en este caso puede ser la función ReLU o tanh.

- **Stride o paso ( $S$ ):** análogo al  $S$  de las capas de *pooling*, pero en este caso los valores que puede tomar son 1 o 2 píxeles de salto o desplazamiento. Con esto se quiere imitar el comportamiento de otras redes clásicas como las ResNet que buscan evitar el sub-muestreo o *subsampling* producido por las capas *pooling*, logrando el mismo efecto al utilizar un *stride* de 2 píxeles que comprime y reduce la imagen a la mitad.

Finalmente, tras decodificar y generar la capa convolucional, se añade una capa de normalización por lotes (*Batch normalization*, Sec. 3.1.2.2), que además de utilizarse como método de regularización, también reduce sustancialmente el tiempo de entrenamiento; además, se añade una capa de *Padding*, también explicada, para conservar el tamaño original de la imagen de entrada tras la convolución (Ec. 4.3.1).

$$P = \frac{K - 1}{2} \quad (4.4)$$

- **Capa Convolucional residual:** capas ya presentadas y explicadas en Sec. 3.1.2.2, caracterizadas por los bloques residuales y las conexiones de salto o atajos (*residual blocks* y *skip connections*) que buscan aumentar la profundidad del modelo sin perder el contexto inicial lo que facilita y mejora los entrenamientos.

Como se observa en la Tabla 4.3.1, los parámetros de configuración para este tipo de capas es exactamente igual que el definido para las capas convolucionales simples. Sin embargo, se deja una puerta abierta al crear este tipo de capas residuales, en concreto para el posterior cierre del bloque que se realiza única y exclusivamente cuando se cumple alguno de los siguientes casos: que se añada una capa de *Pooling*; que a continuación se añada otra capa convolucional residual; o que se haya llegado al final de la codificación, es decir, cuando se haya llegado al límite de bits que conforman un individuo.

Debido a la aleatoriedad que presenta el proceso de creación y generación de este tipo de capas, sobre todo a la hora de cerrar los bloques residuales, es posible que se generen incompatibilidades; en concreto la diferencia de tamaños a la hora de realizar la operación final de suma o *addition*, lo que implica que no se pueda realizar esta última operación y no se cierre el bloque residual. Para dar solución a estas discrepancias, se propone que cuando el tamaño de la imagen  $x$  sea diferente al tamaño de  $x'$ , se genere una convolución equivalente para adaptar el tamaño de ambas y así poder realizar de forma exitosa la operación final de suma o adición como se muestra en la Figura 4.6.

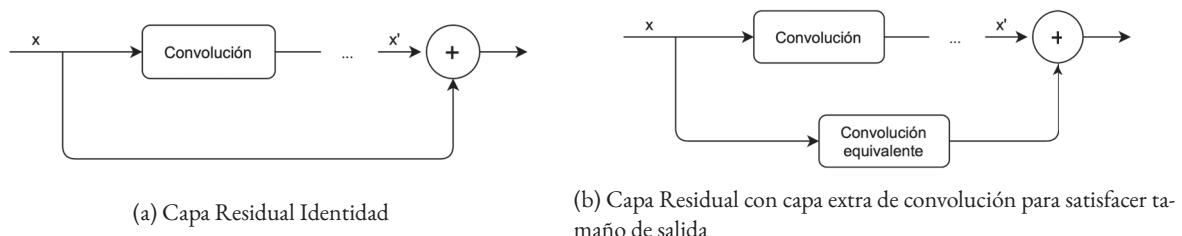


Figura 4.6: Diferencia entre una capa residual básica y otra con una operación convolucional. Imagen de [15]

#### 4.3.2. Evaluación. Cálculo de la *fitness*

La fase de evaluación supone un punto crítico en el proceso evolutivo; este se diseña de forma personalizada en base el problema a resolver. Como ya se ha comentado, este problema busca una red convolucional pequeña, sencilla y con una alta precisión de clasificación, por lo que la población estará formada por individuos que codifican diferentes CNNs. En definitiva, se pueden diferenciar dos objetivos claros:

1. Redes con una **alta capacidad de clasificación**.

## 2. Redes pequeñas y computacionalmente simples y sencillas.

Por eso mismo, se va a trabajar con un valor de *fitness* que definirá la calidad del individuo, es decir, cómo de bien se adapta al problema y si este se acerca a la situación óptima deseada; esta *fitness* estará compuesta por una combinación de valores, uno por cada objetivo a cumplir. Para evaluar estos individuos, primeramente se deben decodificar, obteniendo una arquitectura con una determinada topología que se someterá a un proceso de valoración.

Como se ha visto en el marco teórico, Sec. 3.1.2.2, existen diferentes métricas de rendimiento. Las más comunes son la pérdida y la precisión; para obtener estas métricas es indispensable entrenar la red, así se podrá cuantificar la capacidad de clasificación de la red, y con ello también se obtendrá parte del nivel de cualificación del individuo. En el caso de que el problema fuese uniobjetivo y solo se buscasen redes con una buena capacidad de clasificación o etiquetado, con una de estas métricas se obtendría directamente el valor de la *fitness*.

En cuanto al segundo objetivo, la búsqueda de redes pequeñas y poco complejas, primero se debe definir qué es la complejidad de una red y cómo se cuantifica; existen diferentes estudios que analizan más a fondo este área [247, 248]. En este proyecto se va a priorizar la sencillez de la red; para ello se va a utilizar el tamaño, definido por el número de capas, y la sencillez, computacionalmente hablando, de la red, para ello se buscará que el número de parámetros que participan en el proceso de entrenamiento sea lo más pequeño posible, lo que significa que el número de operaciones también será significativamente bajo.

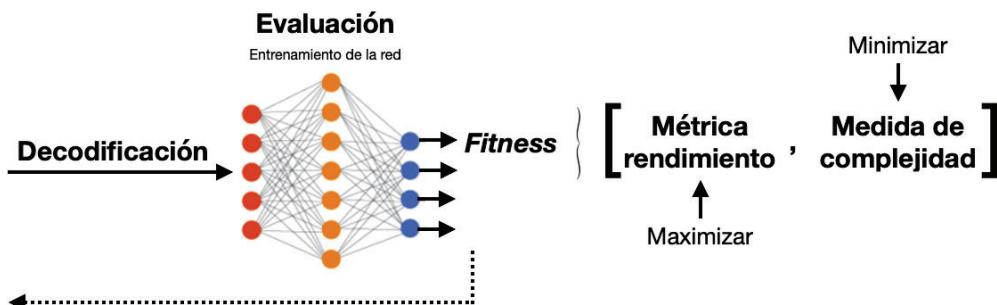


Figura 4.7: *Fitness* del problema multiobjetivo

Los modelos entrenados obtenidos durante el proceso evolutivo, no se evaluarán con el conjunto de datos test hasta haber finalizado el algoritmo evolutivo, y esta evaluación se realizará única y exclusivamente sobre aquellas redes prometedoras, cuya *fitness* se acerca al óptimo global, es decir, se cumplen los requisitos definidos una precisión de clasificación alta y número de parámetros bajo); con esto se busca reducir las operaciones y con ello, agilizar el proceso. También cabe destacar, que por esta misma razón, para agilizar y hacer un uso más eficiente de los recursos, aquellos individuos que se repitan a lo largo de las generaciones, se entrenarán y evaluarán una única vez; ya que estas técnicas conllevan ya de por sí un alto consumo, se intenta agilizar el proceso lo máximo posible.

Operador genético		Tipo	
Inicialización		Aleatoriedad	
Selección		NSGA-II: Selección por torneo	
Cruce	Uniforme	Prob. Individuo: 90 % Prob. Gen: 20 %	
Mutación	Uniforme	Prob. Individuo: 5 % Prob. Gen: 20 %	
Reemplazo		NSGA-II: elitismo + Frente de Pareto + Crowding distance	

Tabla 4.5: Operadores y valores de probabilidad utilizados en los experimentos finales

## 4.4. Estrategias de desarrollo

En este apartado se desarrolla en detalle la línea y flujo de trabajo seguido a lo largo del proyecto, así como la hipótesis inicial.

Como ya se ha explicado en el estado del arte 2, actualmente, la agricultura de precisión se encuentra en auge, ha ganado mucho protagonismo en los últimos años debido a las ventajas socio-económicas y medioambientales que suponen. Este gran desarrollo conlleva un aumento del diseño y creación de nuevas tecnología en este área, potenciando la implantación de sistemas automatizados en el campo.

Como se ha visto en el marco teórico, las redes convolucionales son las redes más utilizadas en problemas de clasificación de imágenes 3.1.2.2; siendo capaces de clasificar de forma eficiente un gran número de imágenes y clases. Para ello, en la actualidad se proponen redes ya pre-entrenadas, es decir, se promueve el uso de la técnica de transferencia de aprendizaje (*transfer learning*) 3.2.1.1. El hecho de trabajar con *datasets* de gran tamaño y un alto número de clases trae consigo un alto coste computacional, ya que para hacer una extracción de información eficaz, el tamaño de las redes se debe incrementar de forma directamente proporcional al número de clases a discernir, aumentando así el número de capas, parámetros, operaciones y transformaciones, y por ende, el coste computacional final. Esta alta demanda de recursos es una limitación para el uso de *edge devices* [249, 250, 251], ya que estos no cuentan con un amplio potencial computacional capaz de soportar los requerimientos de una red y procesos de estas características, así como de la gran cantidad de datos necesarios para su entrenamiento.

Por todo esto, el objetivo final de este proyecto es buscar una forma de obtener CNNs más pequeñas, más rápidas e igual de eficientes y precisas a la hora de clasificar un *dataset* determinado, que otras redes más complejas. Por eso mismo, se explora la opción de optimizar las arquitecturas y su proceso de diseño con ayuda de la computación evolutiva, más específicamente, con los algoritmos genéticos. Estos algoritmos tienen un gran rendimiento en el campo de la optimización de problemas complejos por lo que puede ser una buena herramienta para la resolución del problema de clasificación de malas hierbas; siendo este mecanismo, extrapolable a otros problemas o áreas donde la clasificación de imágenes sea un punto crítico.

Finalmente, en este proyecto se intenta verificar la **hipótesis inicial**, que consiste en la capacidad de diseñar nuevas arquitecturas de CNNs con la ayuda de técnicas de optimización basadas en algoritmos genéticos, donde dichas redes tendrán un tamaño inferior, reduciendo así el número de capas y parámetros, pero sin sacrificar su capacidad de clasificación, consiguiendo CNNs más rápidas y sencillas pero igual o más precisas clasificando, que otras arquitecturas ya existentes.

### 4.4.1. Primera ejecución

Esta primera fase corresponde con la etapa de adaptación y primer contacto con las librerías de desarrollo. Estas pruebas se realizan tanto en local como en el servidor CESGA; los principales ajustes y puntos abordados en esta primera etapa son:

- Código no adaptado para SLURM ni al sistema de colas que utiliza CESGA; se parte de un código desarrollado en local, basado en el trabajo [15]. También es necesario ajustar y revisar el entorno de trabajo (*enviroment*), librerías y paquetes necesarios, versiones, incompatibilidades, etc. Destacar que esta primera etapa se comienza realizando en Finisterrae II pero finalmente, el proyecto se migra a Finisterrae III, con los ajustes y adaptaciones pertinentes.
- El código inicial es aplicable a un problema uniobjetivo, donde solo se toma como *fitness* la precisión de la red durante el entrenamiento. Este enfoque no cuadra con el objetivo de este proyecto, ya que el hecho de contemplar únicamente el *accuracy*, métrica de rendimiento de la red, hace que se evolucione hacia individuos, en este caso redes, cuya capacidad de clasificación sea excelente, sin tener en cuenta el tamaño, tiempo o coste computacional que esto conlleve. Este enfoque puede hacer que se generen arquitecturas extremadamente grandes y complejas, o incluso arquitecturas ya existentes, haciendo que estas sean incompatibles e imposibles de integrar en un sistema de detección y tratamiento en tiempo real.

Primeramente, se ajustan las librerías y paquetes utilizados. CESGA tiene un amplio repertorio de módulos y paquetes ya instalados, pero algunas librerías como DEAP, indispensable para el desarrollo del proyecto, es necesario instalarlas de forma manual; en cambio, otras librerías como *open-cv* (cv2) [252], tampoco disponible en CESGA, se sustituyen por otras semejantes que sí se encuentran en los módulos ya instalados del servidor, como *Pillow* (PIL) [244].

Se desarrolla el código de un AG, basado en el algoritmo evolutivo multiobjetivo NSGA-II (Sec. 3.3.2.1). Como se ha visto en la Sec. 4.3.2, los individuos codifican CNNs, por lo que el entrenamiento de estas redes será la forma de evaluar a cada uno de los individuos de la población, asignándoles una *fitness*. Se comienza tomando como valor de la *fitness* una lista con dos valores, lo que indica que es un problema con dos objetivos (multiobjetivo). Uno de ellos es la métrica de rendimiento de la red, el *accuracy*, ya que se busca una red cuya capacidad de clasificación sea alta; y a su vez, también se busca que la red sea sencilla, por lo que se calcula el número de parámetros entrenables de la red y se utiliza como segundo valor de la *fitness*; únicamente se tiene en cuenta los parámetros que verdaderamente participan en el proceso de entrenamiento, y que por ello, tienen un consumo y demanda de recursos superior (pesos). En el fragmento de código que se muestra a continuación, Cod. 4.4.1, se aprecia cómo la *fitness* devuelve una lista de dos valores, [*accuracy*, *num\_parametros*], y como la función de evaluación, definida como *eval\_fitness*, hace referencia al proceso de entrenamiento de la red, a partir del cual se obtienen los valores que conformarán la *fitness*.

```

1 def evaluate(individual, gen=None, idx=None):
2     #Esta funcion devuelve el valor de la fitness de un individuo
3     if gen is not None: #Generacion
4         GEN = gen
5     else:
6         GEN = current_gen
7     if idx is not None: #id del individuo dentro de la generacion GEN
8         INDIV = idx
9     else:
10        INDIV = 0
11
12    #eval_fitness llama a la funcion de entrenamiento de la red
13    fitness_acc, fitness_params = eval_fitness(individual, GEN, INDIV)
14
15    #Definicion de la fitness como: [accuracy, num_parametros]
16    return fitness_acc, fitness_params

```

En este primer contacto, se realizan pruebas con un número de generaciones, individuos y *epochs* de entrenamiento de las redes, muy reducido; lo que se busca es poder optimizar el comportamiento del algoritmo antes de realizar experimentos más complejos. Este tipo de procesos conllevan un gran uso de recursos y un alto tiempo de ejecución, por lo que para obtener una idea general de cómo está siendo el comportamiento del algoritmo y

si este se ajusta al comportamiento esperado o deseado, se define una situación simple y sencilla que facilite su optimización. Aquí se demuestra cómo de necesario es un entorno de ejecución tan potente como CESGA, ya que estos procesos evolutivos combinados con los entrenamientos de redes convolucionales de clasificación de imágenes requieren de unos recursos computacionales muy altos.

También se realizan pruebas con y sin GPU, demostrando así que el tiempo de procesamiento se reduce considerablemente al utilizar GPUs, necesitando aproximadamente la mitad de tiempo para realizar las mismas generaciones y número de evaluaciones de individuos. Esta afirmación es aproximada, ya que aunque queda demostrado que se reduce el tiempo de ejecución del algoritmo, nunca se evaluarán exactamente los mismos individuos, es decir, nunca se entrenarán las mismas redes, ya que se parte de una población inicial aleatoria; por lo que aún evaluando el mismo número de arquitecturas, estas pueden variar entre una prueba y otra, en cuanto a tamaño, número de parámetros, etc.; aún así, las pruebas se reproducen repetidamente y se demuestra que aunque no es una reducción fija o cuantificable, las pruebas con GPU reducen significativamente el tiempo de ejecución del algoritmo respecto a las pruebas desarrolladas únicamente con CPUs. En la tabla 4.6, se muestran los valores promedio obtenidos tras realizar 10 pruebas de cada una de las situaciones (CPU, 1 GPU y 2 GPUs), demostrando diferencias y aumentos del tiempo de hasta un 6000 %. Para estos experimentos se han reducido el número de individuos de la población a 4 y el número de generaciones se ha definido también como 4; los individuos siguen estando codificados con cadenas binarias de 500 bits, es decir, que representan arquitecturas que pueden contener hasta 50 capas.

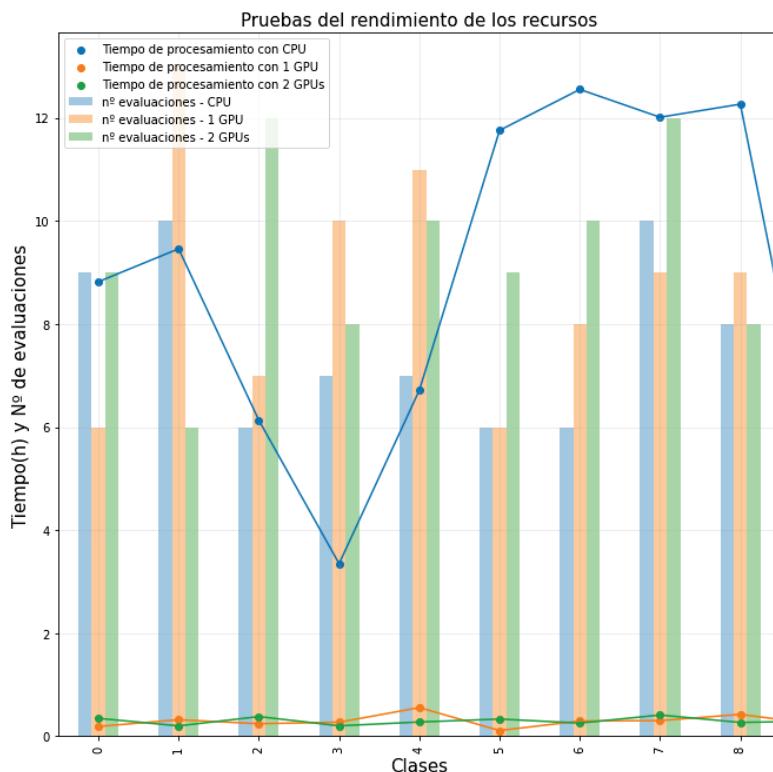


Figura 4.8: Gráfico con los resultados al realizar 10 entrenamientos para evaluar el rendimiento del hardware disponible

CPU		1 GPU		2 GPUs	
Tiempo medio	Tiempo medio por evaluación	Tiempo medio	Tiempo medio por evaluación	Tiempo medio	Tiempo medio por evaluación
8:02:36	<b>1:01:18</b>	0:16:16	<b>0:01:53</b>	0:17:25	<b>0:01:51</b>

Tabla 4.6: Resumen asociado a los experimentos y resultados representados en la Fig. 4.8

En resumen, de esta primera prueba se obtienen las siguientes ideas y mejoras, que se tomarán como punto de partida para la siguiente ejecución:

- Código funcional y compatible con el servidor CESGA.
- Algoritmo multiobjetivo, con dos objetivos, *accuracy* y número de parámetros entrenables.
- Los experimentos con GPU son más rápidos y eficientes que con CPU; pero no hay diferencia entre el uso de 1 y 2 GPUs, lo que significa que el código no contempla esta situación, haciendo un uso inadecuado de los recursos.

#### 4.4.2. Segunda ejecución: entrenamiento distribuido

Partiendo de los resultados obtenidos en la primera ejecución, se ajusta el código para poder hacer un uso eficiente de los recursos. Para ello, se modifica de tal forma que se utilicen las dos GPUs disponibles; por eso mismo se apuesta por realizar un entrenamiento distribuido.

El entrenamiento distribuido consiste en dividir la carga de trabajo que conlleva el entrenamiento de un modelo entre varios procesadores que trabajan en paralelo para acelerar el entrenamiento. En este caso particular, se trabaja en una máquina con múltiples GPUs, es decir, se realiza un entrenamiento síncrono a través de múltiples réplicas en una misma máquina o nodo; esto es posible gracias a la librería de Tensorflow *Mirrored Strategy* [253]. *tf.distribute.Strategy* es una API fácil de usar para distribuir el entrenamiento del modelo en múltiples GPU, TPU y máquinas, mediante la API de alto nivel en TensorFlow y Keras.

Primero se confirma el número de GPUs disponibles; y una vez confirmados los recursos, se puede definir la estrategia a seguir:

```
1 devices = tensorflow.config.experimental.list_physical_devices('GPU')
2
3 strategy = tensorflow.distribute.MirroredStrategy()
```

Al utilizar estrategias de distribución, toda la creación de variables debe realizarse dentro de la estrategia, lo que se conocen como *mirrored variables*; esto duplicará las variables, creando una copia en todas las réplicas para mantenerse sincronizadas. También se debe tener en cuenta al trabajar con *batches*, que el tamaño del *batch* debe definirse de forma conjunta para que este se divida de igual manera en cada una de las réplicas; por ejemplo, si se quiere trabajar con lotes de 64 muestras (*batch\_size* = 64), y se quiere distribuir entre 4 réplicas, el tamaño del *batch* global debe ser *batch\_global* =  $64 * 4 = 256$ . Este mismo concepto de distribución y reparto, se debe aplicar en el *dataset*, distribuyéndose también el conjunto de datos de trabajo entre las réplicas disponibles, como se aprecia en la Fig. 4.9.

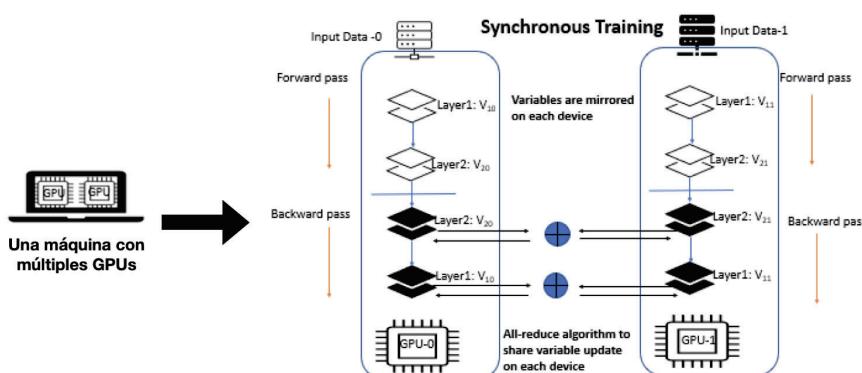


Figura 4.9: Ejemplo gráfico del funcionamiento del entrenamiento distribuido en una máquina con 2 GPUs

Con este nuevo enfoque no solo se aprovechan y utilizan eficientemente los recursos, sino que se reduce sustancialmente el tiempo de procesamiento, agilizando aún más los entrenamientos.

A continuación se enumeran los avances, ideas y cambios realizados en esta segunda ejecución, que darán paso al próximo enfoque, la tercera ejecución:

- Ahorro de tiempo y uso eficiente de los recursos disponibles gracias a la implementación del entrenamiento distribuido. Se ha conseguido distribuir todo el proceso de entrenamiento entre las GPUs disponibles en el nodo de trabajo; sin olvidar la distribución de los datos de entrenamiento, así como las variables que se generan durante el entrenamiento, en definitiva, todos los cálculos, desde convoluciones, cálculos del *loss*, e incluso la actualización de los pesos y proceso de retroalimentación, se han realizado de manera compartida entre dos GPUs, liberando, facilitando y agilizando estas operaciones; esta reducción de carga se ha traducido en una disminución considerable de los tiempos de procesamiento. Sin olvidar un punto esencial ya comentado anteriormente, en relación a las comparaciones que se realizan entre diferentes ejecuciones del algoritmo evolutivo, nunca se evalúan los mismos individuos ya que se parte de individuos creados de forma aleatoria, lo que influye en el resto del flujo evolutivo, haciendo que estas comparaciones sean aproximadas y orientativas, sin poder llegar a ser completamente cuantificables.
- También se modifica la implementación del algoritmo genético, en concreto la parte del código que hace referencia al funcionamiento elitista; inicialmente, se utiliza una de las funciones más utilizadas para algoritmos evolutivos elitistas de la librería *deap*, *deap.tools.HallOfFame*. Esta función identifica al mejor individuo que haya habido en la población durante el proceso evolutivo pero únicamente tiene en cuenta el primer valor de la *fitness*, lo que la hace incompatible con este problema multiobjetivo y con el algoritmo utilizado (NSGA-II). Por eso mismo, se modifica y se sustituye por la función *deap.tools.ParetoFront*, también de *deap*, que a diferencia del anterior, contiene y guarda a todos los individuos no dominados que hayan estado en algún momento en la población.
- Debido a la mejora conseguida con esta modificación, se realizan experimentos más complejos, con un mayor número de generaciones e individuos y entrenamientos completos de hasta 100 *epochs*. El objetivo es ver cómo evoluciona el algoritmo y si estos individuos realmente van mejorando y acercándose al ideal buscado. En este caso, se comprueba que a medida que pasan las generaciones se realizan menos evaluaciones, como ya se ha podido ver en la Tab. 4.6, ya que cuando se obtiene un descendiente (hijo) cuya red que codifica ya ha sido evaluada, este entrenamiento no se repite (Sec. 4.3.2). Esto puede deberse principalmente porque:
  1. Diferentes codificaciones den lugar a un mismo punto en el espacio de búsqueda, lo que significaría que la codificación es mejorable o incluso errónea.
  2. La falta de diversidad se deba a que la población es demasiado pequeña y al aplicar los operadores genéticos no haya margen de actuación, obteniendo individuos que ya han participado anteriormente en alguna generación del ciclo evolutivo.
  3. También puede deberse a una mala elección de la probabilidad de mutación, que esta sea demasiado pequeña y no se aplique con la suficiente frecuencia, afectando directamente al proceso de exploración o diversidad de la población.

### 4.4.3. Tercera ejecución: exploración vs. explotación

Llegados a este punto, el tiempo de procesamiento se ha reducido considerablemente, completando el ciclo evolutivo definido en menos tiempo. También se ha descubierto una reducción del número de evaluaciones, es decir, una disminución de la creación de individuos nuevos con el paso de las generaciones; esta es una señal de alarma, significa que hay falta de diversidad ya que aparecen los mismos individuos y arquitecturas de forma repetida a lo largo de las generaciones evolutivas. Por eso mismo, esta ejecución está orientada a la búsqueda

del equilibrio entre la exploración y la explotación; ambas estrategias están directamente relacionadas con los operadores genéticos de cruce y mutación. El operador de cruce fomenta la búsqueda local o intensificación del espacio de búsqueda (explotación); mientras que la mutación produce cambios más aleatorios, fomentando así la exploración y realizando una búsqueda más amplia y global. Gracias a la exploración, aumenta la diversidad, pudiendo identificar zonas prometedoras que junto con el proceso de explotación, se podrán examinar de forma más profunda e intensa hasta encontrar el óptimo global.

Se analiza la causa de la convergencia prematura detectada en la fase anterior, y se concluye que no es un problema de codificación:

- No hay individuos con cromosomas diferentes que codifiquen una misma red, sino que el espacio de búsqueda se encuentra correctamente representado por la codificación elegida.
- También se analizan las consecuencias que conllevan las disparidades entre cromosomas; cómo y cuánto afecta una simple y pequeña diferencia entre bits, es decir, si cambiar un único bit en el genoma de un individuo genera algún impacto, y si este es considerable o no. Para ello se parte del *individuo\_0*, que puede llegar a tener hasta 50 capas, por lo que la longitud de su cadena será de  $10 * 50 = 500$  bits (ya que una capa se representa con 10 bits, Sec. 4.3.1), y se modifica un único bit, obteniendo el *individuo\_1*; se repite el proceso, obteniendo un segundo individuo, *individuo\_2*. Como se puede ver en la Fig. 4.10, un simple cambio en un bit genera topologías a simple vista diferentes, y en términos computacionales, este cambio también produce diferencias muy notorias, sobre todo en el número de parámetros, llegando a conseguir hasta una reducción superior a los 2 millones de parámetros entre una arquitectura y otra. Como se aprecia en las topologías resultantes, los cambios se producen en las primeras capas, donde el número de entradas, dimensiones y cálculos a realizar son mucho más grandes, de ahí que estas diferencias relacionadas con el número de parámetros puedan llegar a ser tan bruscas; los bits modificados se encuentran en la primera capa, por eso mismo las topologías difieren únicamente al inicio de la misma, siguiendo una misma distribución de las capas finales de la arquitectura (Fig. 4.10).

Una vez demostrada la consistencia de la codificación utilizada, se aborda la segunda posible causa del problema de falta de diversidad, que el tamaño de la población no sea lo suficientemente grande. También se aumenta la longitud de los individuos, es decir, el número de capas que se pueden codificar; como se ha explicado anteriormente, este proceso de optimización se realiza con situaciones irreales y sencillas en términos computacionales; esto se realiza para no incrementar los tiempos de procesamiento y poder lanzar un número de experimentos mayor. Por eso mismo, para comprobar si realmente es el tamaño de la población la causa de esta falta de diversidad en los individuos, se definen cadenas de un tamaño considerablemente grande para que haya un número alto de posibles combinaciones diferentes durante el proceso de reproducción. Si aún con esta posibilidad de generar un número de combinaciones muy alto se siguen repitiendo los individuos a lo largo de las generaciones, esto se traduciría como una convergencia prematura, dejando al algoritmo atrapado o estancado en una zona o punto del espacio de soluciones que no representa la solución global.

Se llega a la conclusión de que el número máximo de capas a codificar, que definen la longitud del cromosoma, tiene cierto peso en el desarrollo de estos procesos, pero no en este problema en particular. Tener una cadena de bits grande, da pie a un número de combinaciones mucho más grande; para el tipo de codificación elegida en este proyecto, si se tienen cadenas de 40 bits, es decir, con un máximo de 4 capas, existen  $2^{40}$  combinaciones, más de 1 billón de combinaciones diferentes; concluyendo que este parámetro no influye directamente en la diversidad de la población, aunque sí tiene un efecto directo en las métricas de rendimiento y clasificación de las redes. Si se acota demasiado este parámetro, el algoritmo estará extremadamente restringido, sin llegar a cubrir completamente el espacio de soluciones; en cambio, si es demasiado grande, puede ser muy difícil o incluso imposible converger en el óptimo global.

En esta ejecución se extraen las siguientes conclusiones y se presentan los avances realizados:

- La falta de diversidad en la población es uno de los problemas más comunes y preocupantes en los algorit-

## Diseño y Metodología

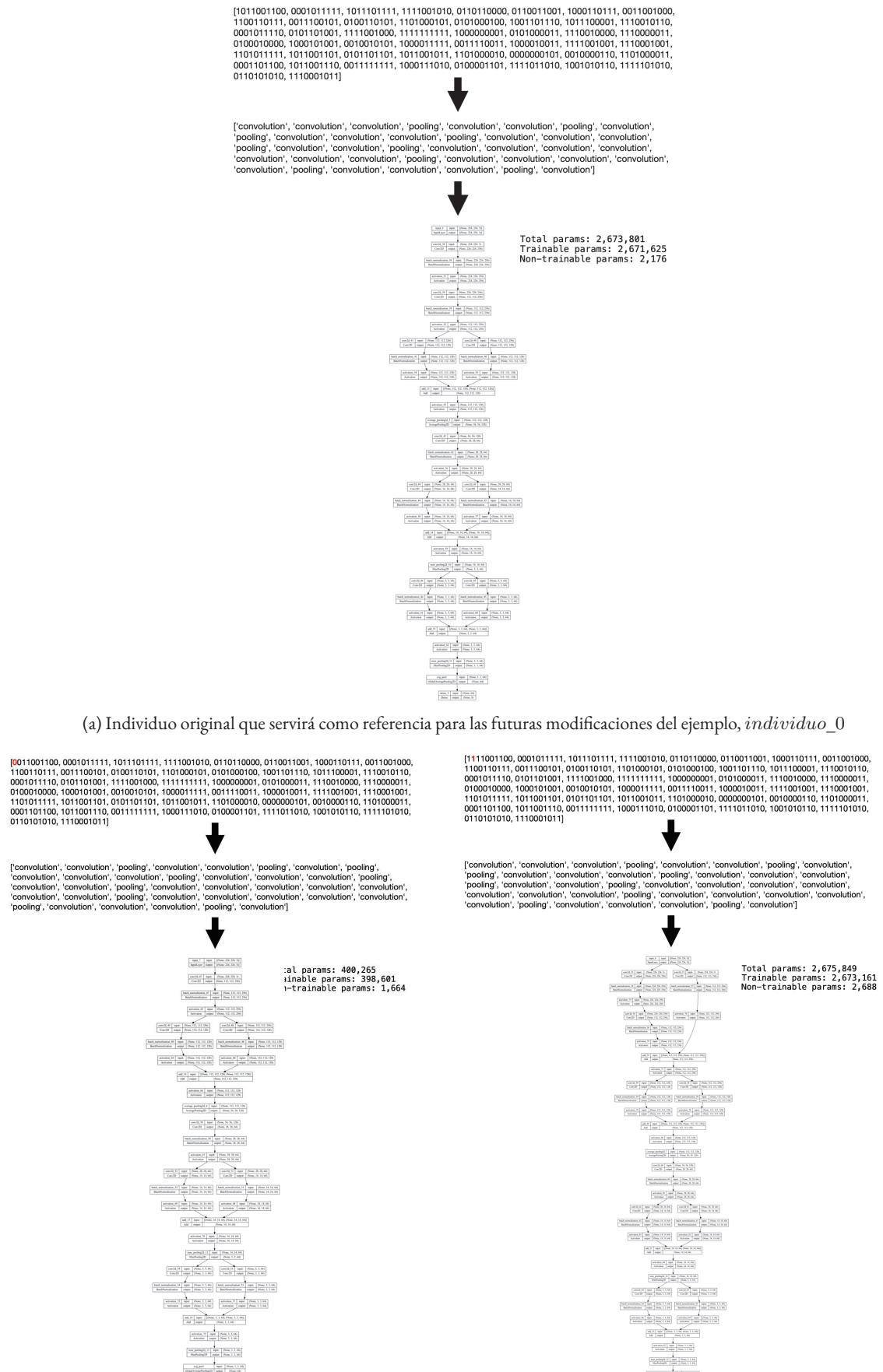


Figura 4.10: Consecuencia de variar un único bit en la cadena o cromosoma de un individuo

mos evolutivos. Se debe encontrar la razón por la que el algoritmo converja en un punto prematuro; los principales motivos que pueden derivar en esta situación son la codificación y la función objetivo, aunque también se deben revisar otros parámetros que dependen del problema y el juicio del desarrollador, como el tamaño de la población o la probabilidad de mutación.

- Queda demostrado que la codificación elegida es correcta, cubre el espacio de búsqueda en su totalidad y da pie a generar un número muy elevado de posibles combinaciones.
- Los resultados obtenidos en esta ejecución son buenos, aunque no los deseados. Se observa cómo el algoritmo antepone la maximización de la precisión de clasificación, frente a la minimización del número de parámetros. Por eso mismo, en la siguiente fase de ajuste del algoritmo se priorizará y buscará el valor de *fitness* más adecuado para este problema concreto.

#### 4.4.4. Cuarta ejecución: ajuste de la *fitness*

Llegados a este punto, en esta última fase se plantean algunas preguntas base que se van a buscar resolver en esta última ejecución:

- La *fitness* elegida, ¿es correcta?
- ¿Existe alguna otra forma para definir esta función de evaluación que sea más acorde a los requisitos definidos y el problema a resolver?
- Si se consigue converger con la función de evaluación definida, ¿convergerá en el punto deseado?
- Para este problema en concreto, ¿cómo se evalúa la capacidad de clasificación de la red?, ¿es la precisión del entrenamiento el valor de *fitness* más adecuado?
- Para este problema en concreto, ¿qué es la complejidad de la red?, ¿está bien representada con el número de parámetros?

En esta última fase, se busca el valor óptimo de la *fitness*, es decir, la mejor forma de evaluar a los individuos en función de los objetivos del proyecto. Para ello se estudian y analizan diferentes estrategias de evaluación que prioricen la búsqueda de redes con arquitecturas más pequeñas y simples, sin dejar a un lado una buena capacidad de clasificación.

##### 4.4.4.1. Precisión vs. Tamaño

Como se ha visto en la fase anterior, el algoritmo antepone las redes con una precisión alta frente a las arquitecturas pequeñas. Esto significa que a medida que aumenten las generaciones el algoritmo seguirá generando arquitecturas más profundas, ya que estas suelen tener una capacidad de extracción de información mayor, convergiendo en un óptimo local no acorde con la finalidad de este proyecto. Como se ha visto en la Sec. 3.3.2.1, los frentes se generan en función de la función de evaluación definida, en este caso particular se busca maximizar la precisión de clasificación y minimizar el tamaño y complejidad de la red.

Durante el proceso de pruebas, se cuestiona si el orden definido dentro de la lista que representa la *fitness* influye o no en la creación de los frentes y en consecuencia, también en el ordenamiento de los individuos. Finalmente, se demuestra de forma empírica que los frentes se mantienen iguales, es decir, que contienen a los mismos individuos, aunque el orden de los mismos sí que varía; esto es así siempre que los frentes se conformen por más de 3 individuos, ya que en otro caso, el frente estaría formado únicamente por uno o ningún punto, afectando drásticamente la definición de la *fitness* a la definición del frente. Por lo que se afirma que el orden definido dentro de la *fitness* influye en el orden de los individuos, es decir, que la selección de los individuos a preservar sí puede llegar a verse afectada por el orden definido dentro de la *fitness*. Finalmente, se concluye que sí influye que la *fitness* sea `[accuracy, num_parmetros]` o `[num_parmetros, accuracy]` (Fig. 4.11).

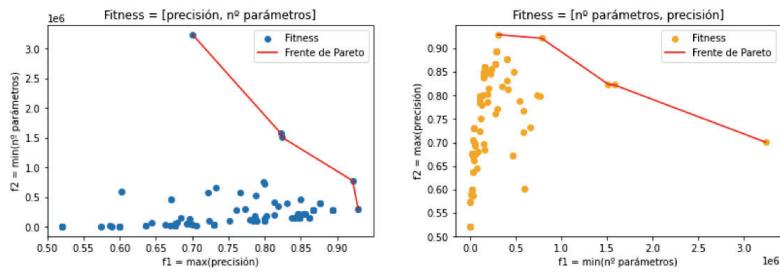


Figura 4.11: Ejemplo gráfico de que el orden de la *fitness* genera los mismos frentes de Pareto, aunque el orden de los individuos sí varía

Una vez demostrada la influencia del orden de definición de la *fitness* multiobjetivo en el proceso evolutivo, concretamente en la selección de individuos, se llega a un momento clave, definir si los valores que componen la *fitness* representan correctamente los objetivos del problema.

### 4.4.4.2. Capacidad de clasificación de la red

Como se ha comentado, se busca una red que sea capaz de resolver el problema de clasificación de malas hierbas con una precisión alta, es decir, con un alto porcentaje de aciertos. Por eso mismo, se decide utilizar la métrica de rendimiento de entrenamiento *accuracy*. La precisión o *accuracy* es una métrica que describe y cuantifica el rendimiento del modelo en todas las clases, muy útil cuando todas las clases tienen la misma importancia, como se da en este caso. Se calcula como la relación entre el número de predicciones correctas y el número total de predicciones (Ec. 4.4.4.2):

$$Accuracy = \frac{\text{Predicciones\_correctas}}{\text{Predicciones\_correctas} + \text{Predicciones\_incorrectas}} \quad (4.5)$$

Se decide no evaluar todos los modelos durante el proceso de evaluación ya que se invierten muchos recursos y se incrementa el tiempo de procesamiento del algoritmo; únicamente se evalúan aquellas redes que han resultado del proceso evolutivo como los mejores individuos. Por eso mismo, se comienza utilizando la precisión obtenida durante el entrenamiento como valor de evaluación, pero se comprueba que esta no es una medida lo suficientemente fiable. A lo largo de las generaciones, si la *fitness* está correctamente definida según los requisitos del problema, las redes convergerán en arquitecturas mucho más simples y pequeñas, cuyos entrenamientos puedan llegar a verse afectadas por el fenómeno *overfitting*; se han definido diferentes técnicas de regularización para intentar paliar este fenómeno pero esto puede no ser suficiente. Por eso mismo, se decide seleccionar el *accuracy* de validación, también calculado durante el proceso de entrenamiento. Este es un buen indicador de *overfitting* y se asemeja más a los resultados que se pueden obtener con el dataset de test (Fig. 4.12).

Solucionando así el problema del sobre-entrenamiento, se descarta el uso de la pérdida como valor de *fitness*, ya que esta métrica no cuantifica de forma directa la calidad de las predicciones, sino que es una medida orientativa que se utiliza durante el entrenamiento.

### 4.4.4.3. Complejidad de una red

En este apartado se analizan diferentes formas de definir la complejidad de la red [254]:

- **Número de parámetros:** hace referencia a todos los parámetros que participan en el proceso de entrenamiento, incluidos los pesos y sesgos; estos parámetros también se denominan parámetros entrenables. Para calcular y cuantificar estos parámetros se hace uso del módulo de Tensorflow, *model.summary()* [255].

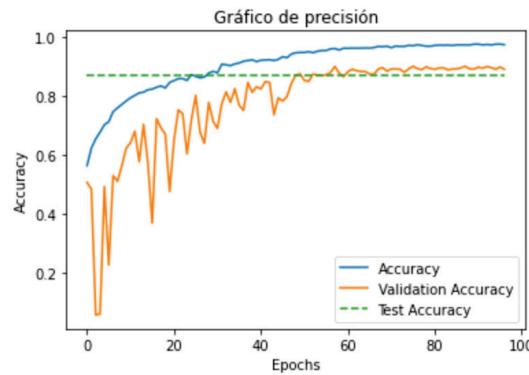


Figura 4.12: Ejemplo del entrenamiento de una red obtenida durante el proceso evolutivo

- **Número de capas:** para calcular el número de capas de las arquitecturas se utiliza Tensorflow, `model.layers()`. Inicialmente se trabaja con la totalidad de capas, aunque se ha de destacar que las capas que conllevan un uso más exhaustivo de los recursos y mayor coste computacional, son las capas convolucionales y las totalmente conectadas.
- **Otros hiperparámetros:** también existen otros hiperparámetros que influyen en el tiempo de procesamiento final y la demanda de recursos (Fig. 4.13).

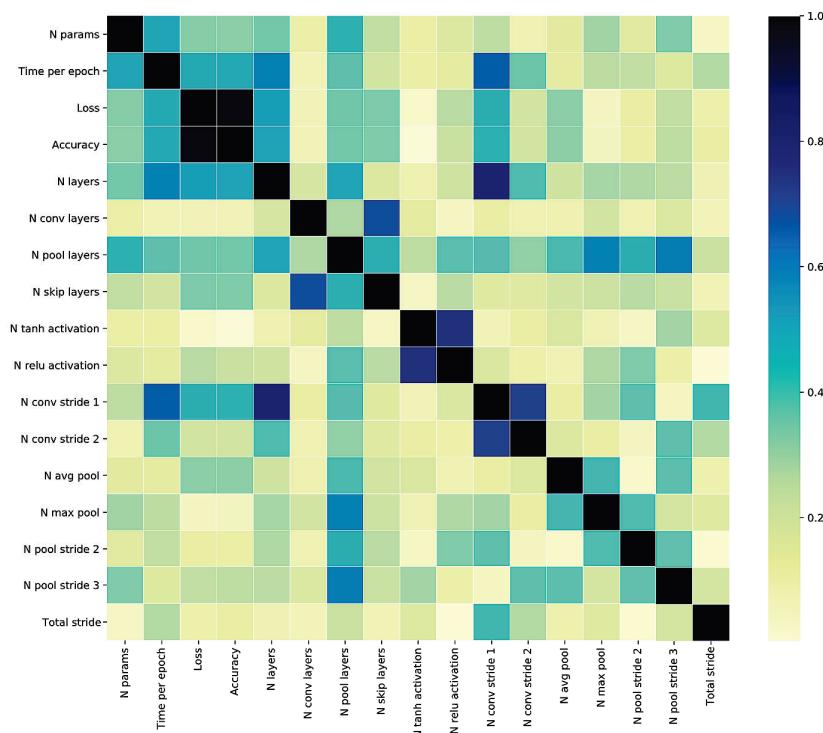


Figura 4.13: Correlación de Spearman entre las variables de una arquitectura CNN. Imagen de [15]

Como existen numerosos parámetros capaces de definir la complejidad de la red, en este proyecto se trabaja con el número de parámetros y el número de capas como primera aproximación. Estos parámetros se pueden interpretar de forma sencilla y directa, y se pueden comparar fácilmente con otras arquitecturas ya existentes.

## Capítulo 5

# Resultados y Discusiones

Como se ha visto en la Sec. 4.4, se ha seguido un flujo de trabajo progresivo, de menos a más, donde se han ejecutado numerosas pruebas y experimentos hasta llegar a la configuración que mejor se adapta a este problema.

Se comprueba que el valor de la fitness que mejor se adapta a este proyecto y con el que es más sencillo y fácil trabajar es [ $nº\_capas$ ,  $accuracy$ ]. Esto es principalmente porque el número de capas es una cifra más sencilla y reducida, mucho más fácil de manejar que el número de parámetros y que a simple vista ofrece mucha información; aunque como se ha comentado en la Sec. 4.4.4.3, existen otras posibilidades de limitar aún más este parámetro, como es el hecho de identificar y reducir únicamente las capas que llevan un coste mayor (CNN o FC); esto no se aborda en este primer estudio, sino que se propone como una futura modificación.

El primer enfoque que se da sobre estos experimentos finales, consiste en reducir y limitar el número de capas a generar, ya que esto coincide con el fin último del problema, encontrar redes pequeñas. Esta idea se ve sustentada por lo comentado en la Sec. 4.4.4.1, donde se demuestra numéricamente que debido al tipo de codificación diseñada, el número de combinaciones posibles sigue siendo muy elevado y este hecho no tiene por qué perjudicar la diversidad del sistema. Sin embargo, como se ve en la Fig. 5.2, este hiperparámetro influye fuertemente en el desarrollo completo del algoritmo evolutivo. Los procesos evolutivos necesitan de un número elevado de generaciones para converger, aún más cuando se trabaja con problemas complejos; por eso mismo, se han realizado tantos experimentos como el servidor ha permitido, debido a las limitaciones temporales que tiene, y se ha decidido predecir cuál puede o podría ser el comportamiento de estos algoritmos si el proceso evolutivo se hubiese alargado en el tiempo, es decir, si se hubiese aumentado el número de generaciones. Para predecir o estimar los resultado de las próximas generaciones de algunos de los experimentos realizados, se ha partido de los datos disponibles (generaciones realizadas), y se les ha aplicado una regresión lineal para entrenar y predecir cómo sería su evolución en un futuro hipotético (Fig. 5.2); para ello se ha utilizado *LinearRegression* de Scikit-learn [256].

En estos ejemplos se puede ver claramente como definiendo un número de capas  $X$ , se pueden llegar a generar arquitecturas de tamaños superiores. Esto se debe a las estructuras residuales y a las capas que componen el bloque de clasificación, es decir, las capas finales de la arquitectura, ya que estas se encuentran fijas o predefinidas, variando únicamente el bloque de extracción de información.

También cabe destacar que el límite que define un correcto funcionamiento del proceso evolutivo, según los criterios y requisitos de este problema en concreto, se encuentra en la codificación de 20 capas (individuos con cadenas de 200 bits). Observando las rectas de regresión y sus predicciones, a partir de las 20 capas codificadas se consigue el objetivo, evolucionar hacia redes más pequeñas (disminución del  $nº$  de capas) y más precisas (aumento del  $accuracy$ ). En cambio, por debajo de las 20 capas codificadas, se limita tanto la definición de los individuos y arquitecturas, convergiendo hacia redes más pequeñas (disminución del número de capas) pero menos pre-

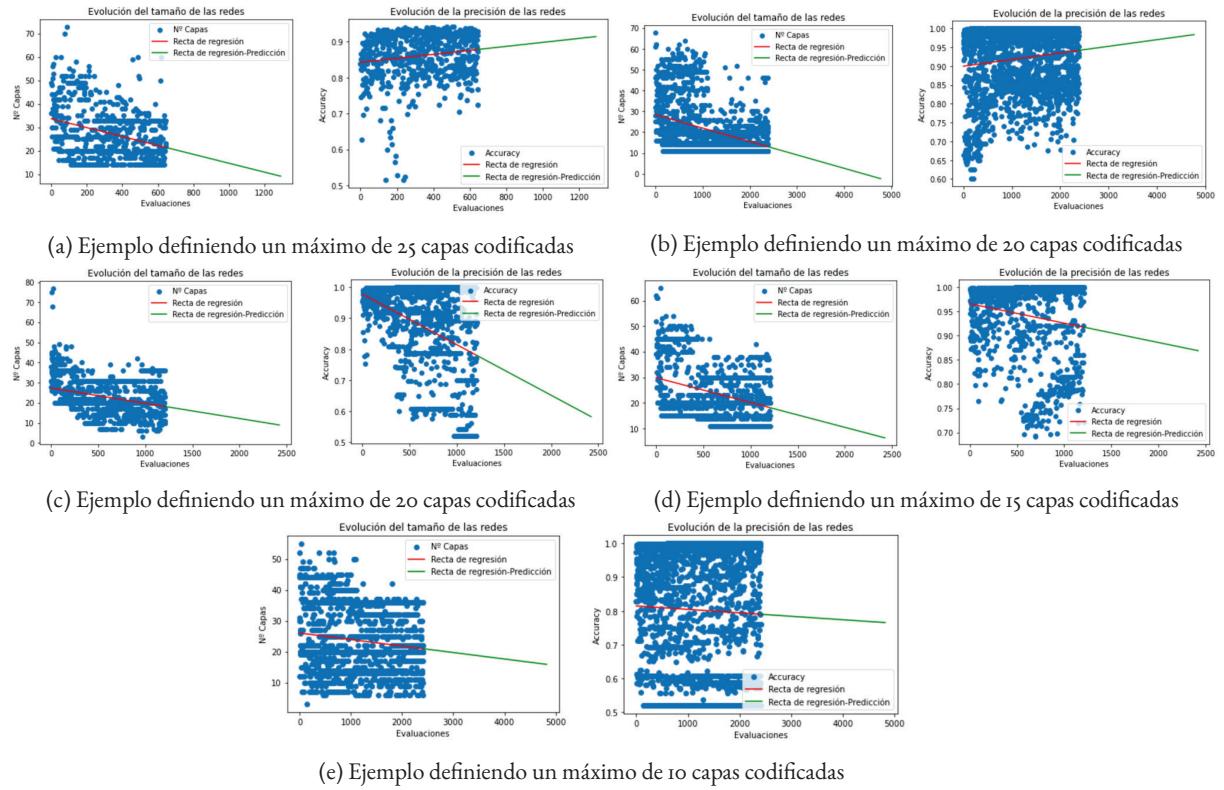


Figura 5.1: Ejemplo de la evolución del flujo evolutivo según el tamaño (izq.) y la precisión de las redes (dch.); en función del número de capas máximo a codificar

cisas (disminución del *accuracy*). Con esto se concluye que como bien se había intuido, este parámetro limita el proceso evolutivo, por lo que es conveniente definir un número de capas a codificar relativamente alto y que sea el propio algoritmo el que a través de la *fitness* definida sea capaz de generar poblaciones que evolucionen hacia el objetivo último, pudiendo así alcanzar el verdadero óptimo global. Para estos experimentos no se llega a definir este parámetro por encima de 25 capas, ya que esto ralentizaría de forma excesiva los entrenamientos.

También se puede apreciar en las gráficas (Fig. 5.2) el porqué de la dificultad de llevar a cabo estos procesos. En el eje de abscisas se representa el número de evaluaciones realizadas, o lo que es lo mismo, el número de entrenamientos ejecutados para el desarrollo del algoritmo genético de cada uno de estos experimentos, llegando a realizar entre 1000 y 3000 evaluaciones. También se destaca que estos entrenamientos no se caracterizan por su simplicidad y rapidez, sino que se llegan a evaluar arquitecturas muy grandes y complejas, con un alto coste computacional. Por lo que una vez más, queda demostrado que estos experimentos se han podido realizar gracias a los recursos disponibles en CESGA y su alta capacidad de cómputo.

Gracias a la aplicación del algoritmo NSGA-II, como resultado de ejecutar estos procesos evolutivos se obtiene un conjunto de individuos que cumplen con los requisitos del problema y se encuentran ordenados de mejor a peor. Gracias a la codificación presentada y su capacidad de generar arquitecturas tan variadas, junto con el enfoque elitista del NSGA-II, no existe una arquitectura única que resuelva el problema, sino que se presentan redes diferentes que representan diferentes puntos del espacio de soluciones y que se ajustan en diferente medida a cada uno de los objetivos. Gracias al NSGA-II se pueden obtener todos los individuos que mejor cumplen con los objetivos definidos en la *fitness*, pudiendo elegir aquél que más se adapte a las necesidades. Con esto se quiere destacar las posibilidades que presenta este algoritmo y su alta capacidad de adaptación, ya que por ejemplo, en un caso práctico donde se busca integrar una de estas redes en un sistema de hardware limitado, se puede elegir una red generada por el algoritmo, que saque el máximo partido de las capacidades y recursos del dispositivo y cuya precisión de clasificación sea la más alta posible; en cambio, si este dispositivo tiene unas

## Resultados y Discusiones

---

prestaciones inferiores, se podría elegir una red más pequeña pero que a su vez tenga un *accuracy* máximo; en el caso de que no se tuvieran restricciones o limitaciones computacionales, se podría elegir aquella red con un mayor *accuracy*, asegurándonos de que a su vez tiene un nivel de complejidad mínimo. Otro aspecto a destacar, es que se devuelven modelos entrenados, por lo que estos se pueden utilizar directamente para procesos de clasificación, o como parte de un entrenamiento diferente en el que se aplica transferencia del conocimiento.

A continuación se presentan algunas de las mejores arquitecturas generadas con el algoritmo, así como sus características principales, métricas y resultados obtenidos:

Nombre	Nº de capas	Nº de parámetros	Accuracy			Loss		
			Train	Val	Test	Train	Val	Test
Inception-v3	159	$23 * 10^6$		95.1 %			-	
ResNet50	50	$25 * 10^6$		95.7 %			-	
<b><u>ECNN-o.1</u></b>	11	$0.81 * 10^6$	89.79 %	82.35 %	76.96 %	0.3689	0.5324	0.6632
ECNN-o.2	14 (Gen-160)	$3.24 * 10^6$	99.81 %	72.98 %	70.67 %	0.0449	0.9090	0.9340
ECNN-o.3	14 (Gen-161)	$3.24 * 10^6$	95.20 %	81.83 %	75.73 %	0.2150	0.5383	0.6671
ECNN-o.4	14 (Gen-197)	$0.54 * 10^6$	89.49 %	85.03 %	81.64 %	0.3557	0.4576	0.5423
ECNN-o.5	15	$1.28 * 10^6$	99.67 %	76.69 %	74.96 %	0.0643	0.6823	0.8091
ECNN-o.6	18	$1.28 * 10^6$	99.67 %	77.21 %	76.81 %	0.0461	0.7837	0.8091
ECNN-o.7	19	$2.51 * 10^6$	99.46 %	77.44 %	72.10 %	0.0469	0.7752	0.9590
ECNN-o.8	20	$1.31 * 10^6$	98.40 %	75.55 %	71.79 %	0.1263	0.7443	0.8209
ECNN-o.9	25	$2.95 * 10^6$	98.80 %	83.49 %	79.10 %	0.0919	0.5003	0.6424
<b><u>ECNN-i.1</u></b>	14	$0.29 * 10^6$	86.83 %	83.27 %	73.67 %	0.4098	0.5172	0.7405
ECNN-i.2	16	$0.69 * 10^6$	90.20 %	85.09 %	71.87 %	0.3351	0.4730	0.8566
ECNN-i.3	17	$0.48 * 10^6$	96.24 %	87.95 %	80.24 %	0.1428	0.3745	0.5517
ECNN-i.4	18	$0.60 * 10^6$	94.82 %	89.32 %	85.44 %	0.1843	0.3259	0.4238
ECNN-i.5	19	$0.61 * 10^6$	95.22 %	89.61 %	78.41 %	0.1802	0.3315	0.6964
ECNN-i.6	20	$0.66 * 10^6$	97.76 %	90.23 %	86.98 %	0.0897	0.3056	0.3759
ECNN-i.7	22	$0.85 * 10^6$	98.43 %	91.55 %	83.84 %	0.0728	0.2875	0.5087
<b><u>ECNN-i.8</u></b>	24	$0.77 * 10^6$	97.87 %	92.40 %	89.05 %	0.0878	0.2491	0.3174
ECNN-i.9	25	$0.90 * 10^6$	97.70 %	93.55 %	80.41 %	0.0890	0.2066	0.6227
ECNN-i.10	33	$0.85 * 10^6$	98.18 %	94.12 %	87.18 %	0.0734	0.2192	0.3948

Tabla 5.1: Algunas de las redes generadas con el algoritmo evolutivo y sus métricas y características más representativas

Como se puede ver, sí existe una disminución sustancial en los tamaños de las redes generadas respecto las clásicas analizadas en [14], lo que conlleva una reducción de la complejidad de las redes, del tiempo de procesamiento y de su coste computacional. En cambio, la capacidad de clasificación de estas redes evolutivas no llegan a alcanzar los niveles de estas arquitecturas, algo que no sorprende debido al limitado número de generaciones y de individuos con los que se ha trabajado; esta situación, como se ha visto con las predicciones representadas en la Fig. 5.2, se podría corregir o incluso invertir si se dejase al algoritmo evolucionar durante más tiempo. Cabe destacar que aun así, se han conseguido valores de precisión muy altos para el tipo de problema que se tiene, ya que como se ha explicado en 4.1.2, el problema de clasificación de malas hierbas no es un problema trivial, sino que estas imágenes son difíciles de distinguir y etiquetar hasta para el ojo humano.

A partir de los resultados mostrados en la Tab. 5.1, se puede concluir que:

- Hay una gran diferencia entre los resultados obtenidos con las redes del experimento 0 (ECNN-o.n) y las del experimento 1 (ECNN-i.n). En el primer experimento se ejecutan 200 generaciones, con una población de 12 individuos, y un máximo de 20 capas para codificar; este entrenamiento corresponde con la Fig. 5.1b. En referencia al experimento 1, se ejecutan 55 generaciones, con una población de 12 indi-

viduos, y un máximo de 25 capas para codificar; este entrenamiento corresponde con la Fig. 5.1a. Este parámetro se ha reducido para disminuir el tiempo de procesamiento, ya que si se define como se comentó inicialmente como 50 capas, se podrían generar arquitecturas mucho más grandes con un mayor consumo de recursos y tiempo.

- Como suele pasar en los entrenamientos de redes, los resultados obtenidos durante el entrenamiento no son reales. Los valores de estas métricas disminuyen cuando se evalúan los modelos con datos desconocidos para la red, como son las imágenes del *dataset* de test. En términos generales, esto se cumple con los valores obtenidos, lo que puede significar que no hay sobreajuste; esto se confirmará más adelante con las representaciones gráficas.
- En el experimento 0, se destaca la evaluación de tres arquitecturas diferentes con el mismo número de capas, los experimentos ECNN-0.2, el ECNN-0.3 y el ECNN-0.4. En concreto la red ECNN-0.2, y ECNN-0.3, tienen el mismo número de capas y de parámetros entrenables; estas capas únicamente se diferencian por la última capa convolucional que se aplica en el bloque de extracción, donde la ECNN-0.2 aplica un kernel con un *stride* = 2, disminuyendo así las dimensiones de la capa de salida que será el *input* para las siguientes capas, que no generan parámetros, lo que supone que estos sean iguales. Aún así, esta modificación tan simple se traduce en una mejora en la precisión del modelo. También se puede ver cómo el algoritmo a medida que pasan las generaciones se acerca más al objetivo del problema, generando arquitecturas más pequeñas y precisas; como es el caso de la ECNN-04 en comparación con los experimentos más prematuros, ECNN-02 Y ECNN-03.

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
conv2d (Conv2D)	(None, 112, 112, 256)	11776
batch_normalization (BatchN ormalization)	(None, 112, 112, 256)	1024
activation (Activation)	(None, 112, 112, 256)	0
max_pooling2d (MaxPooling2D)	(None, 37, 37, 256)	0
conv2d_1 (Conv2D)	(None, 19, 19, 256)	3211520
batch_normalization_1 (BatchN ormalization)	(None, 19, 19, 256)	1024
activation_1 (Activation)	(None, 19, 19, 256)	0
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 256)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	16448
batch_normalization_2 (BatchN ormalization)	(None, 3, 3, 64)	256
activation_2 (Activation)	(None, 3, 3, 64)	0
avg_pool (GlobalAveragePooling2D)	(None, 64)	0
dense (Dense)	(None, 9)	585
<hr/>		
Total params:	3,242,633	
Trainable params:	3,241,481	
Non-trainable params:	1,152	

(a) Topología de la red ECNN-02

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
conv2d (Conv2D)	(None, 112, 112, 256)	11776
batch_normalization (BatchN ormalization)	(None, 112, 112, 256)	1024
activation (Activation)	(None, 112, 112, 256)	0
max_pooling2d (MaxPooling2D)	(None, 37, 37, 256)	0
conv2d_1 (Conv2D)	(None, 19, 19, 256)	3211520
batch_normalization_1 (BatchN ormalization)	(None, 19, 19, 256)	1024
activation_1 (Activation)	(None, 19, 19, 256)	0
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 256)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	16448
batch_normalization_2 (BatchN ormalization)	(None, 3, 3, 64)	256
activation_2 (Activation)	(None, 3, 3, 64)	0
avg_pool (GlobalAveragePooling2D)	(None, 64)	0
dense (Dense)	(None, 9)	585
<hr/>		
Total params:	3,242,633	
Trainable params:	3,241,481	
Non-trainable params:	1,152	

(b) Topología de la red ECNN-03

Figura 5.2: Comparación de las topologías de los experimentos ECNN-02 y ECNN-03

## Resultados y Discusiones

A continuación, se van a analizar más exhaustivamente las tres arquitecturas que cumplen las siguientes características:

- Mayor precisión en test:** la red que ha conseguido una mayor precisión de clasificación ha sido la ECNN-1.8, logrando un 89 % de precisión en test.

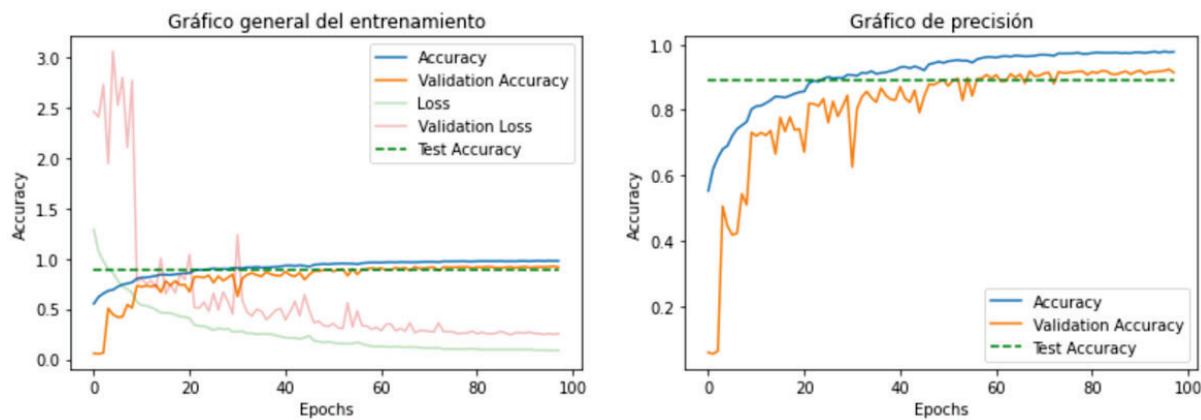
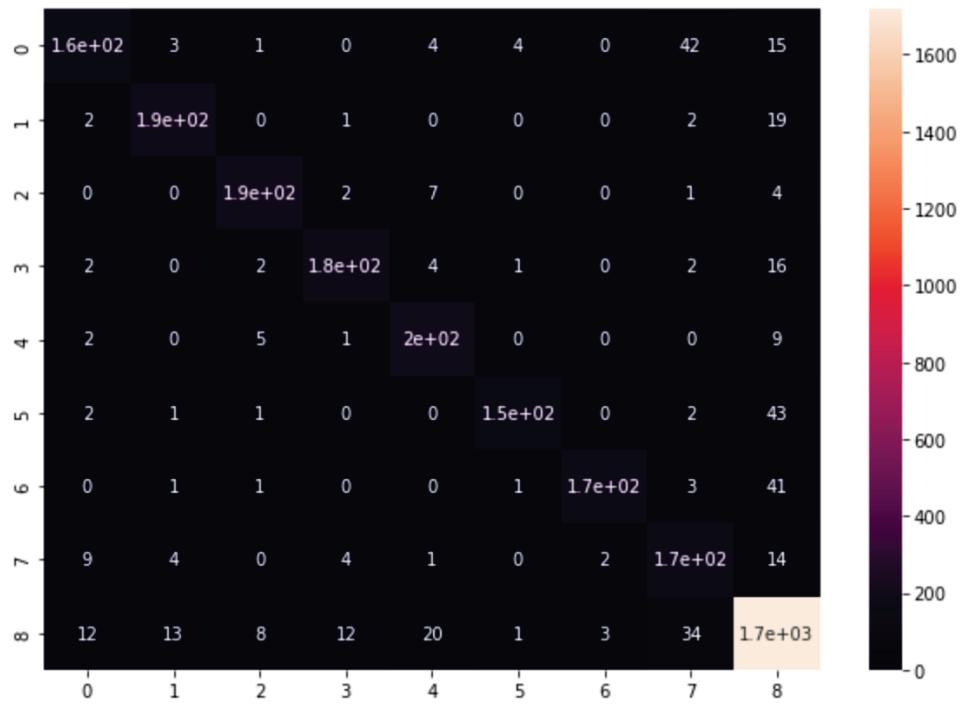


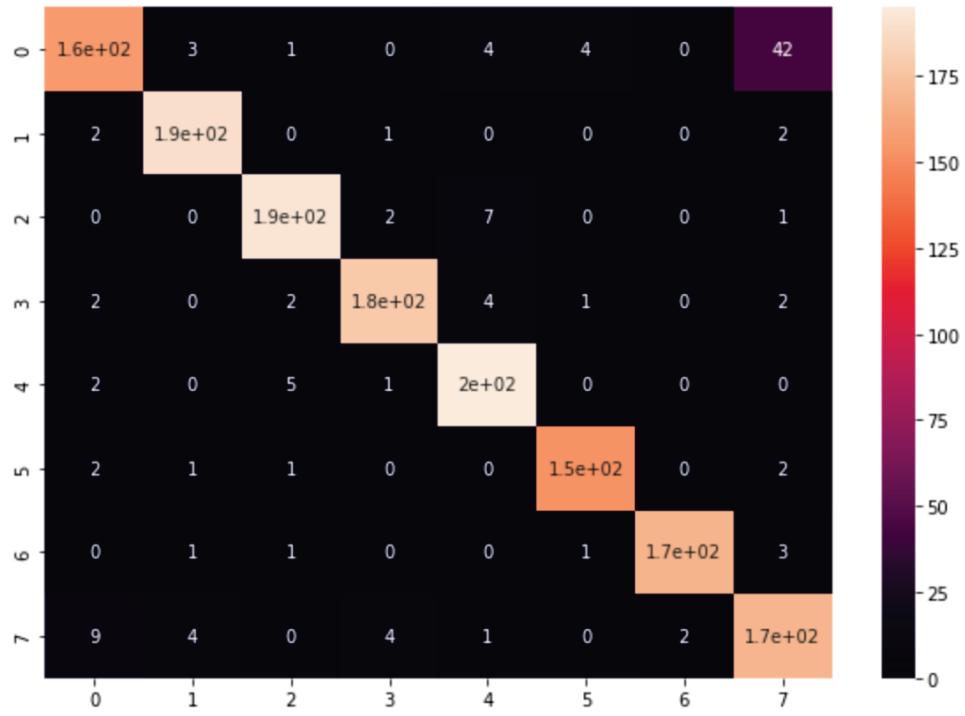
Figura 5.3: Gráfico del entrenamiento y las métricas principales de la ECNN-18

Clase	<i>precision</i>	<i>recall</i>	<i>f1-score</i>	<i>support</i>
Chinee apple	0.84	0.69	0.76	225
Lantana	0.90	0.89	0.89	213
Parkinsonia	0.91	0.93	0.92	206
Parthenium	0.90	0.87	0.88	205
Prickly acacia	0.84	0.92	0.88	212
Rubber vine	0.96	0.76	0.85	202
Siam weed	0.97	0.78	0.87	215
Snake weed	0.66	0.83	0.74	203
Negative	0.91	0.94	0.93	1821
<b>accuracy</b>			<b>0.89</b>	3502
<b>macro avg</b>	0.88	0.85	0.86	3502
<b>weighted avg</b>	0.89	0.89	0.89	3502

Tabla 5.2: Informe de clasificación de la red ECNN-1.8



(a) Matriz de correlación con 9 clases



(b) Matriz de correlación con 8 clases

Figura 5.4: Matriz de correlación de la red ECNN-1.8

## Resultados y Discusiones

2. Menor número de capas: la red más pequeña que se ha conseguido ha sido la ECNN-o.1, con solo 11 capas.

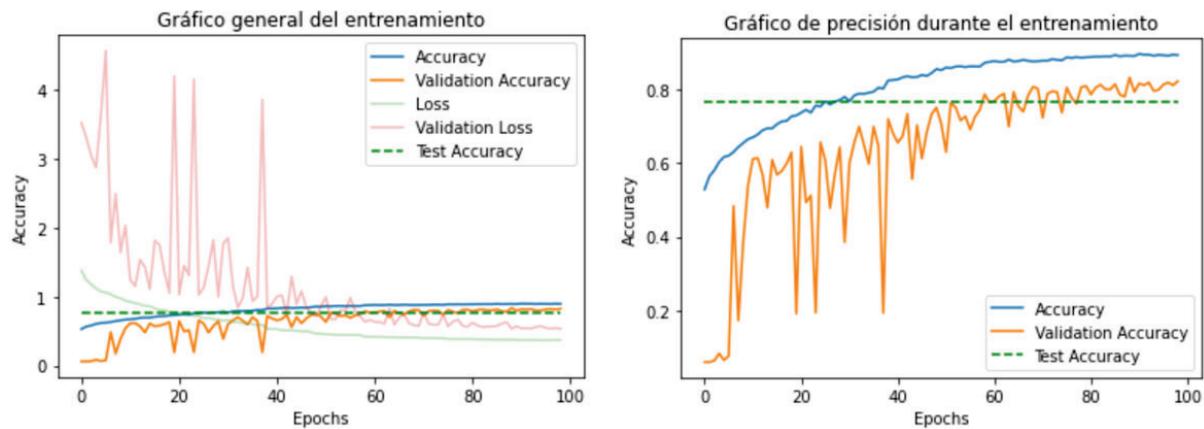
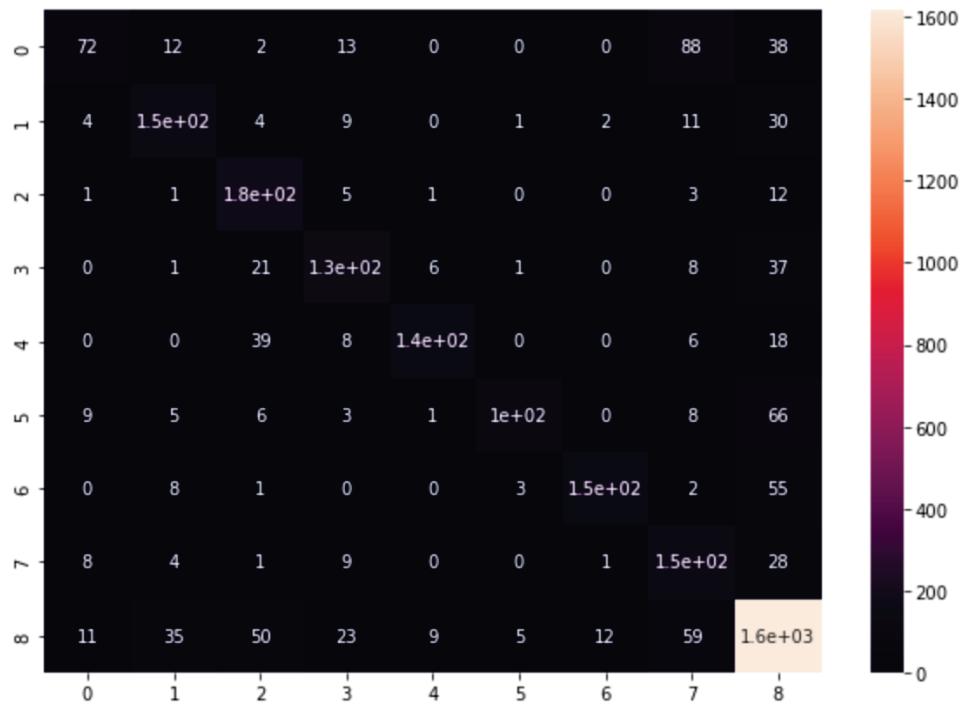


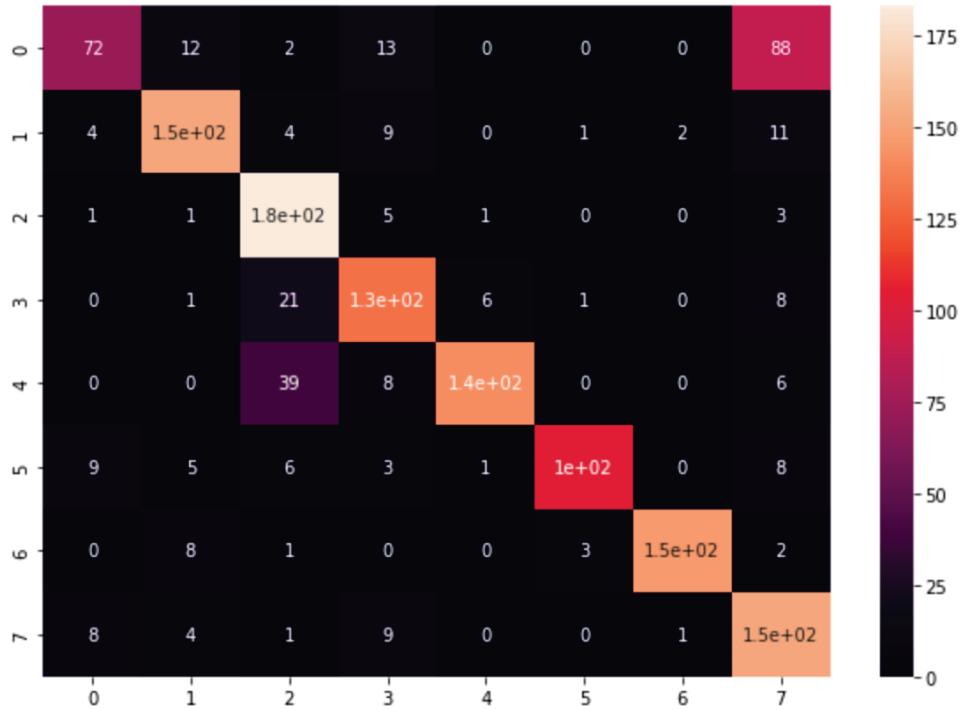
Figura 5.5: Gráfico del entrenamiento y las métricas principales de la ECNN-o.1

Clase	precision	recall	f1-score	support
Chinee apple	0.69	0.32	0.44	225
Lantana	0.70	0.71	0.71	213
Parkinsonia	0.60	0.89	0.71	206
Parthenium	0.65	0.64	0.65	205
Prickly acacia	0.89	0.67	0.76	212
Rubber vine	0.91	0.51	0.66	202
Siam weed	0.91	0.68	0.78	215
Snake weed	0.45	0.75	0.56	203
Negative	0.85	0.89	0.87	1821
<b>accuracy</b>			<b>0.77</b>	3502
<b>macro avg</b>	0.74	0.67	0.68	3502
<b>weighted avg</b>	0.79	0.77	0.77	3502

Tabla 5.3: Informe de clasificación de la red ECNN-o.1



(a) Matriz de correlación con 9 clases



(b) Matriz de correlación con 8 clases

Figura 5.6: Matriz de correlación de la red ECNN-o.I

## Resultados y Discusiones

3. **Menor número de parámetros:** la red con menos carga computacional, es decir, con menos parámetros entrenables es la ECNN-1.1, con tan solo 293065 parámetros ( $0.29 * 10^6$  parámetros).

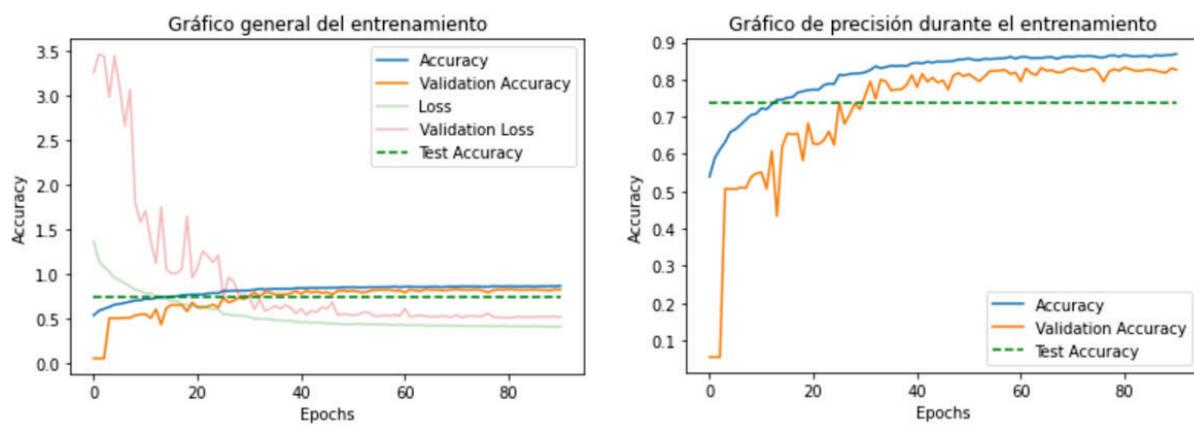
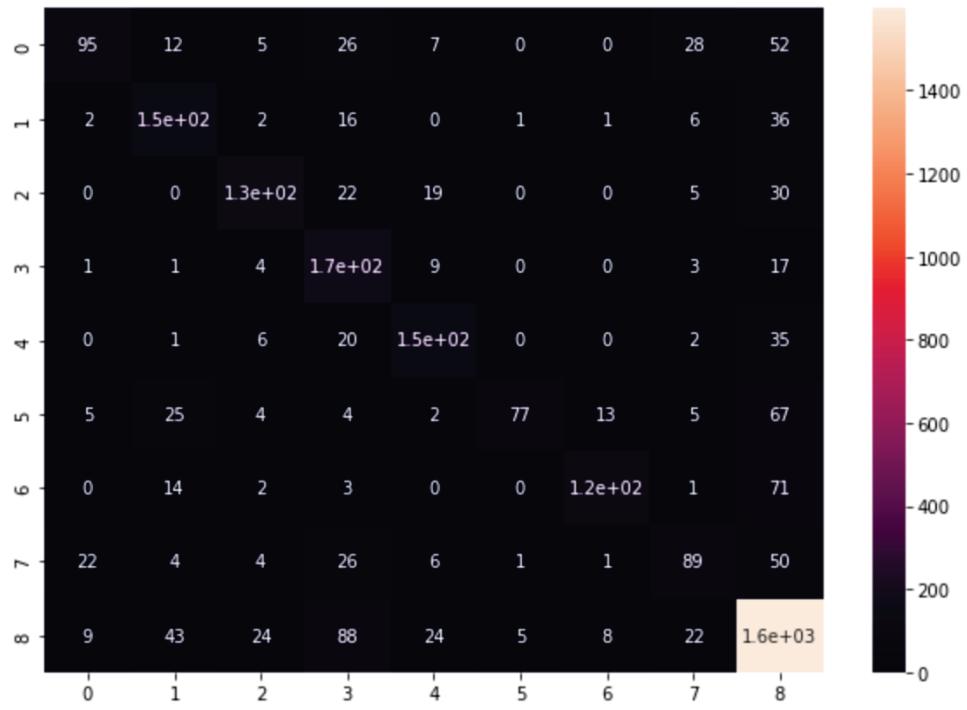


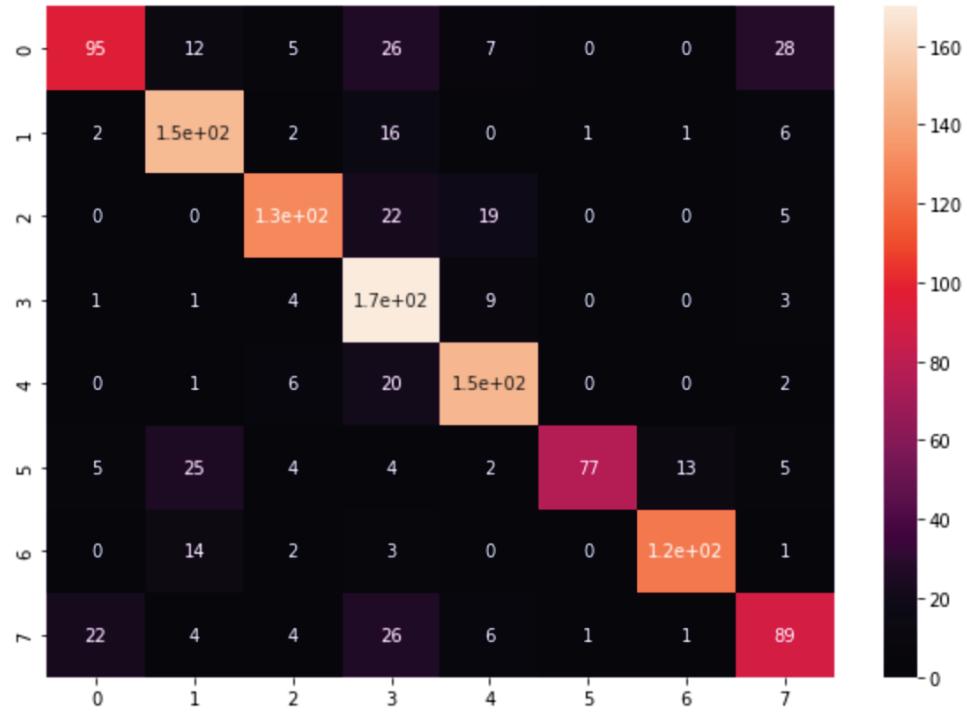
Figura 5.7: Gráfico del entrenamiento y las métricas principales de la ECNN-1.1

Clase	<i>precision</i>	<i>recall</i>	<i>f1-score</i>	<i>support</i>
Chinee apple	0.71	0.42	0.53	225
Lantana	0.60	0.70	0.65	213
Parkinsonia	0.72	0.63	0.67	206
Parthenium	0.45	0.83	0.59	205
Prickly acacia	0.69	0.70	0.69	212
Rubber vine	0.92	0.38	0.54	202
Siam weed	0.84	0.58	0.69	215
Snake weed	0.55	0.44	0.49	203
Negative	0.82	0.88	0.85	1821
<b>accuracy</b>			<b>0.74</b>	3502
<b>macro avg</b>	0.70	0.62	0.63	3502
<b>weighted avg</b>	0.75	0.74	0.73	3502

Tabla 5.4: Informe de clasificación de la red ECNN-1.1



(a) Matriz de correlación con 9 clases



(b) Matriz de correlación con 8 clases

Figura 5.8: Matriz de correlación de la red ECNN-1.1

## Resultados y Discusiones

---

De estos gráficos y resultados se deducen y confirman los siguientes puntos:

- Se confirma que los ajustes de regularización aplicados durante el entrenamiento han sido una buena solución para evitar el sobreentrenamiento; las curvas de entrenamiento siguen una trayectoria coherente y acorde a lo esperado.
- Todas las gráficas de entrenamiento siguen una estructura parecida, prácticamente igual; las curvas de precisión aumentan progresivamente, mientras que la pérdida disminuye; siendo los valores obtenidos durante el proceso de validación peores que los conseguidos durante el entrenamiento. Por esto mismo, para complementar estos datos, se generan las matrices de correlación de cada red; para facilitar la comprensión de las representaciones, se decide construir dos matrices, una con nueve clases y otra eliminando esta novena clase "*negative*", ya que como se ha visto en la Sec. 4.1.2, esta es una clase mayoritaria que descompensaría la representación de la matriz como se puede ver en los ejemplos anteriores (Fig. 5.4, 5.6, 5.8).
- A simple vista, la matriz de correlación y métricas obtenidas con la ECNN-1.8, son bastante superiores a los obtenidos por las otras dos redes. Destacando que toda la diagonal de la matriz se representa con la franja de colores que hace referencia a las cifras más altas que representan el número de aciertos o predicciones realizadas para cada clase, lo que significa que esta red predice o etiqueta con una precisión alta a la mayoría de las clases, algo que también se puede comprobar en la tabla resumen Tab. 5.2. La situación ideal, estaría representada con el valor máximo en la diagonal, siendo el resto de celdas iguales a 0, lo que significaría que no hay predicciones erróneas o cruzadas entre clases. A diferencia de este gráfico, las obtenidas por la ECNN-1.8 y la ECNN-0.1 no son tan precisas, lo que se traduce en una representación de la matriz más irregular. En el caso de la ECNN-0.1, la clase que peor etiqueta y que por tanto le cuesta más identificar, llegando a confundirla con otras clases, es la especie "*Chinee apple*"; mientras que para la ECNN-1.1, a parte de mostrar problemas de clasificación con la clase 0, tampoco etiqueta correctamente las especies "*Rubber vine*" "*Snake weed*", etiquetándolas la mayoría de veces como si estas fuesen "*negative*".
- Estos tres primeros puntos u observaciones anteriores, hacen referencia a la precisión de la red, por lo que a continuación se analizan los resultados obtenidos desde el punto de vista de la complejidad de los modelos; en concreto, a partir de su tamaño, con el número de capas y número de parámetros entrenables.

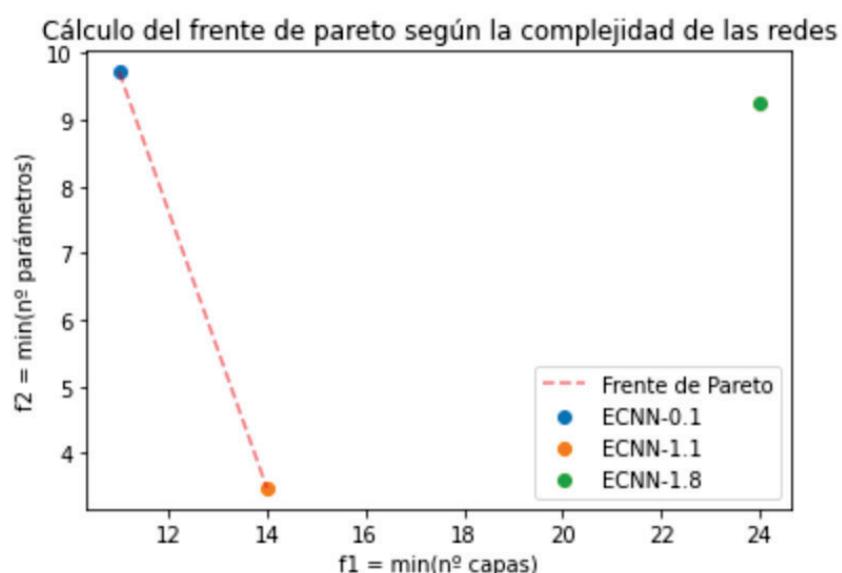


Figura 5.9: Frente de Pareto de los datos: relación entre el nº de capas y el nº de parámetros en las redes

A partir de la representación Fig. 5.9, se ve como la única solución dominada es la que hace referencia a la ECNN-1.8; esto quiere decir, que respecto a cada uno de los objetivos, tanto el número de capas como el de parámetros, existe otra solución con un mejor resultado, en este caso, con un resultado menor. En cuanto a la ECNN-0.1, se confirma que es una solución no dominada, ya que ninguna de las otras dos redes tiene un número menor de capas; de forma análoga, la ECNN-1.1 es una solución no dominada debido al número de parámetros de la red. Como esto no ayuda a la elección de una solución final, se decide calcular la media de parámetros por capa, obteniendo:

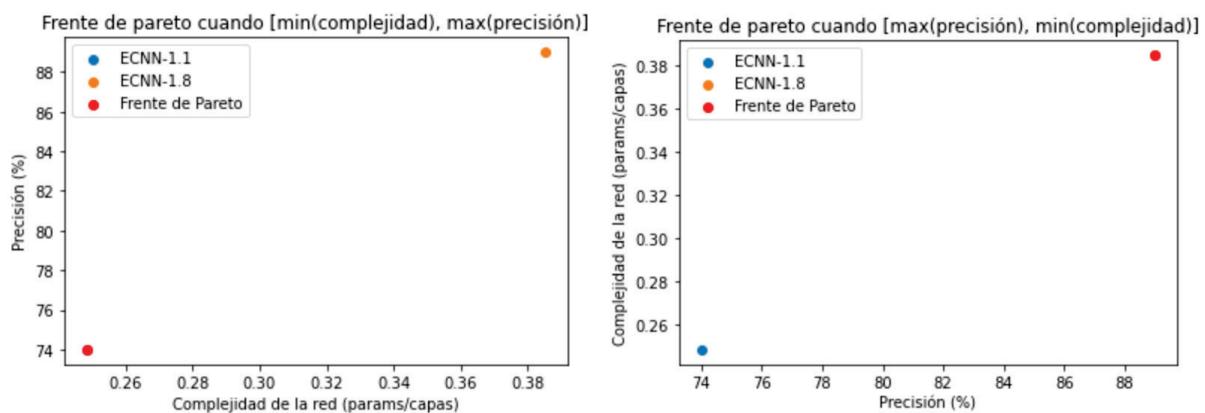
Red	nº capas	nº parámetros	parámetros/capa
ECNN-0.1	11	$0.81 * 10^6$	$7.4 * 10^4$
ECNN-1.1	14	$0.29 * 10^6$	$2.1 * 10^4$
ECNN-1.8	24	$0.77 * 10^6$	$3.2 * 10^4$

Tabla 5.5: Complejidad de las redes

Se confirma que desde un punto de vista de la complejidad de la red, valorada o medida a partir del número de capas y parámetros, la red más simple es la ECNN-1.1.

- Al evaluar los dos objetivos por separado, se han obtenido dos redes solución que maximizan cada uno de estos aspectos. Por un lado la ECNN-1.8 maximiza la precisión de clasificación, y la ECNN-1.1 maximiza la simplicidad de la red, o lo que es lo mismo, minimiza su complejidad.

¿Pero qué ocurre si se evalúan estos objetivos conjuntamente? Como se ve en la Fig. 5.10, ambas soluciones son no-dominadas, por lo que dependiendo de la definición de los objetivos, en este caso de su orden, se dará preferencia a un frente o al otro; como ya se había demostrado en la Sec. 4.4.4.1. Por lo que si el hardware es lo suficientemente potente como para almacenar y ejecutar la red más compleja de entre las soluciones finales (ECNN-1.8), se buscaría maximizar la precisión por lo que la red que mejor se adaptaría a este caso concreto sería la ECNN-1.8. Pero en el caso de que el hardware fuese muy limitado, la solución óptima sería la red más sencilla, la ECNN-1.1.



(a) Caso 1- Frente de Pareto: [min(complejidad), max(precisión)] (b) Caso 2- Frente de Pareto: [max(precisión), min(complejidad)]

Figura 5.10: Frente de Pareto con las dos redes solución

- Finalmente, se afirma que el algoritmo genético que se ha diseñado en este proyecto funciona, y que tras los experimentos realizados las redes solución obtenidas son la ECNN-1.1 y la ECNN-1.8; obteniendo una solución por cada objetivo.

## Capítulo 6

# Conclusiones y Trabajos futuros

La motivación de este trabajo se centra en la verificación de la hipótesis inicial (Sec. 4.4), que afirma que es posible aplicar la computación evolutiva para la generación de redes pequeñas, sencillas y con una alta capacidad de clasificación. En este caso concreto, esta demostración lleva implícito un efecto positivo y de mejora sobre el proceso de clasificación y localización de maleza o malas hierbas en cultivos, para su posterior tratamiento aplicando técnicas de agricultura de precisión, consiguiendo así un mayor rendimiento de los cultivos y disminución de los costes económicos, sociales y medioambientales.

En este proyecto se ha buscado una alternativa a los métodos actuales y clásicos de clasificación de imágenes que aplican redes genéticas y pre-entrenadas, y cuyo uso y aplicación se ve limitado por su alto grado de complejidad y coste computacional. En concreto, en este trabajo se presenta un método novedoso, centrado en el diseño y generación de redes optimizadas, en concreto de CNNs, mucho más sencillas y precisas, y listas para ser aplicadas a problemas de clasificación de imágenes; estas redes se generan de forma automática, siguiendo un proceso de optimización multiobjetivo basado en la estrategia y enfoque de los procesos evolutivos, concretamente, utilizando el algoritmo genético NSGA-II. Esta metodología se caracteriza por ser extrapolable a diferentes casos de estudio donde se trabajen imágenes digitales, o problemas que necesiten del uso de CNNs.

Con este método se han conseguido redes entrenadas para el problema concreto, con un poder de clasificación de prácticamente un 90 % de precisión, y una simplicidad increíblemente notoria respecto algunas arquitecturas populares como la ResNet-50 e Inception v3; comparando la complejidad de las redes generadas y las clásicas a partir del número de capas y parámetros, se ha conseguido una reducción de capas de hasta un 93 % respecto Inception-v3, y del 22 % con respecto a ResNet-50; en referencia a la cantidad de parámetros entrenables de los modelos, estos se han conseguido reducir hasta un 99 %.

Por tanto, se concluye que la hipótesis inicial es válida. Basándonos en los resultados obtenidos, se confirma que se ha conseguido encontrar, con ayuda de los procesos evolutivos, arquitecturas que se adaptan de forma más eficiente al problema de clasificación de especies de malas hierbas.

Aunque los resultados obtenidos son muy prometedores y cubren los objetivos propuestos inicialmente, aún queda mucho potencial por extraer de esta metodología y desarrollo realizado. Para ello se proponen **posibles líneas de trabajo futuro**:

- **Aumentar el número de generaciones.** Para ello se debe modificar el código para poder ejecutarse de forma no continua, sin verse condicionado por las limitaciones de tiempo de uso de los recursos en el servidor, y evitando que el proceso evolutivo se quede en la superficie, y así poder aprovechar todas las ventajas que ofrecen las estrategias evolutivas.
- Gran parte de los hiperparámetros de estas redes se han definido de forma constante o genérica, como el

---

ratio de aprendizaje, el *batch size*, la división del *dataset*, las funciones de activación, etc. Por lo que también se propone **darle una segunda vuelta a la codificación**, incluyendo algunos de estos **hiperparámetros de la red** para conseguir que estas sean aún más diversas y eficientes. También podría permitirse **codificar otros tipos de conexiones entre capas**.

- **Utilizar y probar la aplicación de las redes generadas con otros conjuntos de datos** que contengan las mismas especies de malezas. Así se podría comprobar la capacidad de generalización de los modelos, etiquetando imágenes diferentes, de localizaciones desconocidas y capturadas con métodos y herramientas diferentes.
- **Aplicar y utilizar los modelos generados** para mejorar y facilitar el entrenamiento de otras redes y otros conjuntos de datos. Es decir, utilizar los modelos y arquitecturas generadas y entrenadas durante el proceso evolutivo como material para entrenar otros modelos, participando directamente en el proceso de *transfer learning*.
- Estudiar la posibilidad de **mejorar el valor de la fitness, o incluso aumentar el número de objetivos del problema**. Debido a la dificultad que supone definir la complejidad de la red con un único valor, se propone investigar otras opciones donde se combinen diferentes características, pasando de un problema multiobjetivo con dos objetivos, a tener tres o incluso más objetivos a cumplir.
- **Sacar el mayor partido posible de los recursos ofrecidos por CESGA**. CESGA es un centro de supercomputación por lo que se dispone de hardware muy potente para ejecutar los experimentos, lo que agilizaría de forma considerable los entrenamientos y cálculos, por eso mismo, una de las propuestas que se hace es conseguir **paralelizar el cálculo de la fitness**. En este proyecto se ha trabajado en un único nodo, con dos GPUs que han participado de forma conjunta a la hora de entrenar las redes; CESGA da la posibilidad de disponer varios nodos, por lo que en vez de evaluar un individuos por nodo como se está haciendo hasta el momento, se podría considerar evaluar diferentes individuos de forma paralela, entrenando diferentes arquitecturas en diferentes nodos.
- **Integrar las redes obtenidas en un sistema de detección y tratamiento de precisión en tiempo real**.

# Bibliografía

- [1] X. Wu, S. Aravecchia, P. Lottes, C. Stachniss, and C. Pradalier, “Robotic weed control using automated weed and crop classification,” *Journal of Field Robotics*, vol. 37, no. 2, pp. 322–340, 2020.
- [2] F. Chollet, *Deep learning with Python*. Simon and Schuster, 2021.
- [3] C. Bielza and P. Larrañaga, *Data-driven computational neuroscience: machine learning and statistical models*. Cambridge University Press, 2020.
- [4] A. Morera Munt and J. T. Alcalá Nalvaiz, “Introducción a los modelos de redes neuronales artificiales el perceptrón simple y multicapa,” Ph.D. dissertation, Tesis de pregrado, Universidad Saragoza]. Saguan Repositorio Institucional ..., 2018.
- [5] *The Neural Network Zoo*. The Asimov Institute 2019 [online] Available at: <https://www.asimovinstitute.org/neural-network-zoo/> [accessed Mar. 28, 2022].
- [6] *Multi-layer Perceptron using Keras on MNIST dataset for Digit Classification | by Rana singh | Analytics Vidhya | Medium*. (n.d.). Medium 2021 [online] Available at: <https://medium.com/analytics-vidhya/multi-layer-perceptron-using- keras-on-mnist-dataset-for-digit-classification- problem-relu-a276cbf05e97> [accessed Mar. 28, 2022].
- [7] F. Li, H. Tang, S. Shang, K. Mathiak, and F. Cong, “Classification of heart sounds using convolutional neural network,” *Applied Sciences*, vol. 10, no. 11, p. 3956, 2020.
- [8] “VGGNet model. The structure of the VGGNet model”, ResearchGate. 20, 2021. [https://www.researchgate.net/figure/VGGNet-model-The-structure-of-the-VGGNet-model-is-described-as-follows-The-input-of-fig1\\_346745042](https://www.researchgate.net/figure/VGGNet-model-The-structure-of-the-VGGNet-model-is-described-as-follows-The-input-of-fig1_346745042) [accessed Jul. 07, 2022].
- [9] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [10] “Setting the learning rate of your network”, Jeremy Jordan. 20, 2021. <https://www.jeremyjordan.me/nn-learning-rate/> [accessed Jun. 14, 2022].
- [11] “Nesterov Momentum”, Wiki Golden. 20, 2021. [https://golden.com/wiki/Nesterov\\_momentum](https://golden.com/wiki/Nesterov_momentum) [accessed Jun. 14, 2022].
- [12] Cesga - Centro de Supercomputación de Galicia. 2022. *Computación - Cesga - Centro de Supercomputación de Galicia*. [online] Available at: <https://www.cesga.es/infraestructuras/computacion/> [Accessed 7 June 2022].
- [13] Cesga - Centro de Supercomputación de Galicia. 2022. *Inicio - Cesga - Centro de Supercomputación de Galicia*. [online] Available at: <https://www.cesga.es> [Accessed 7 June 2022].

- [14] A. Olsen, D. A. Konovalov, B. Philippa, P. Ridd, J. C. Wood, J. Johns, W. Banks, B. Grgenti, O. Kenny, J. Whinney *et al.*, “Deepweeds: A multiclass weed species image dataset for deep learning,” *Scientific reports*, vol. 9, no. 1, pp. 1–12, 2019.
- [15] S. Martínez Hamdoun, “Estrategia evolutiva para diseñar redes neuronales convolucionales,” Ph.D. dissertation, Industriales, 2021.
- [16] A. O. Bula, “Importancia de la agricultura en el desarrollo socio-económico,” 2020.
- [17] M. Mazoyer and L. Roudart, *A history of world agriculture: from the neolithic age to the current crisis*. NYU Press, 2006.
- [18] M. B. Tauger, *Agriculture in world history*. Routledge, 2010.
- [19] L. Rickards and S. M. Howden, “Transformational adaptation: agriculture and climate change,” *Crop and Pasture Science*, vol. 63, no. 3, pp. 240–250, 2012.
- [20] M. S. Kang and S. S. Banga, “Global agriculture and climate change,” *Journal of Crop Improvement*, vol. 27, no. 6, pp. 667–692, 2013.
- [21] M. Abdalla and A. Khalil Ibrahim, “The impact of using wireless sensors technology methods on precision agriculture: A review.”
- [22] *Precision Ag Definition*. ISPA, 2020.. [online] Available at: <https://www.ispag.org/about/definition> [accessed Jul. 6, 2022].
- [23] U. Shafi, R. Mumtaz, J. García-Nieto, S. A. Hassan, S. A. R. Zaidi, and N. Iqbal, “Precision agriculture techniques and practices: From considerations to applications,” *Sensors*, vol. 19, no. 17, p. 3796, 2019.
- [24] D. C. Tsouros, S. Bibi, and P. G. Sarigiannidis, “A review on uav-based applications for precision agriculture,” *Information*, vol. 10, no. 11, p. 349, 2019.
- [25] R. P. Sishodia, R. L. Ray, and S. K. Singh, “Applications of remote sensing in precision agriculture: A review,” *Remote Sensing*, vol. 12, no. 19, p. 3136, 2020.
- [26] N. Abu, W. Bukhari, C. Ong, A. Kassim, T. Izzuddin, M. Sukhaimie, M. Norasikin, and A. Rasid, “Internet of things applications in precision agriculture: A review,” *Journal of Robotics and Control (JRC)*, vol. 3, no. 3, pp. 338–347, 2022.
- [27] S. Condran, M. Bewong, M. Z. Islam, L. Maphosa, and L. Zheng, “Machine learning in precision agriculture: A survey on trends, applications and evaluations over two decades,” *IEEE Access*, 2022.
- [28] V. Banthia and G. Chaudaki, “The study on use of artificial intelligence in agriculture,” *Journal of Advanced Research in Applied Artificial Intelligence and Neural Network*, vol. 5, no. 2, pp. 18–22, 2022.
- [29] R. Bongiovanni and J. Lowenberg-DeBoer, “Precision agriculture and sustainability,” *Precision agriculture*, vol. 5, no. 4, pp. 359–387, 2004.
- [30] B. Sahu, S. Chatterjee, S. Mukherjee, and C. Sharma, “Tools of precision agriculture: A review,” *Int. J. Chem. Stud.*, vol. 7, pp. 2692–2696, 2019.
- [31] D. Wei, C. Liping, M. Zhijun, W. Guangwei, and Z. Ruirui, “Review of non-chemical weed management for green agriculture,” *International Journal of Agricultural and Biological Engineering*, vol. 3, no. 4, pp. 52–60, 2010.
- [32] H. Abouzien and W. Haggag, “Weed control in clean agriculture: a review,” *Planta daninha*, vol. 34, pp. 377–392, 2016.

- [33] J. Machleb, G. G. Peteinatos, B. L. Kollenda, D. Andújar, and R. Gerhards, “Sensor-based mechanical weed control: Present state and prospects,” *Computers and electronics in agriculture*, vol. 176, p. 105638, 2020.
- [34] E. Pannacci, M. Farneselli, M. Guiducci, and F. Tei, “Mechanical weed control in onion seed production,” *Crop protection*, vol. 135, p. 105221, 2020.
- [35] I. Gazoulis, P. Kanatas, and N. Antonopoulos, “Cultural practices and mechanical weed control for the management of a low-diversity weed community in spinach,” *Diversity*, vol. 13, no. 12, p. 616, 2021.
- [36] R. A. Andrade, R. S. de Brito, R. F. Mendes, and R. d. C. A. Neto, “Cultural treatments in pineapple crop. management for high yield—review,” *Scientific Electronic Archives*, vol. 14, no. 12, 2021.
- [37] P. Scheepens, H. Müller-Schärer, and C. Kempenaar, “Opportunities for biological weed control in europe,” *BioControl*, vol. 46, no. 2, pp. 127–138, 2001.
- [38] G. Sary, H. El-Naggar, M. Kabesh, M. El-Kramany, G. S. H. Bakhoum *et al.*, “Effect of bio-organic fertilization and some weed control treatments on yield and yield components of wheat,” *World J. Agric. Sci*, vol. 5, no. 1, pp. 55–62, 2009.
- [39] J. Fu, S. Zou, M. Coleman, X. Li, W. Hu, A. Wang, P. Zhang, Z. Zeng, C. Ding, B. Xi *et al.*, “Is it necessary to apply chemical weed control in short-rotation poplar plantations on deep soil sites?” *Industrial Crops and Products*, vol. 184, p. 115025, 2022.
- [40] R. Kumar, P. Gupta, and S. Singh, “Efficacy of chemical herbicides on weed management in onion (*Allium cepa*),” *SCIENTISTS JOINED AS LIFE MEMBER OF SOCIETY OF KRISHI VIGYAN*, p. 112, 2022.
- [41] R. Idziak, H. Waligóra, and V. Szuba, “The influence of agronomical and chemical weed control on weeds of corn,” *Journal of Plant Protection Research*, vol. 62, no. 2, p. 216, 2022.
- [42] B. Ayana, “Efficacy of herbicides to various weeds in perennial crops,” *American Journal of Chemical and Biochemical Engineering*, vol. 6, no. 1, pp. 1–5, 2022.
- [43] R. Gerhards, D. Andújar Sanchez, P. Hamouz, G. G. Peteinatos, S. Christensen, and C. Fernandez-Quintanilla, “Advances in site-specific weed management in agriculture—a review,” *Weed Research*, vol. 62, no. 2, pp. 123–133, 2022.
- [44] R. Raja, T. T. Nguyen, D. C. Slaughter, and S. A. Fennimore, “Real-time weed-crop classification and localisation technique for robotic weed control in lettuce,” *Biosystems Engineering*, vol. 192, pp. 257–274, 2020.
- [45] M. Weis, C. Gutjahr, V. Rueda Ayala, R. Gerhards, C. Ritter, and F. Schölderle, “Precision farming for weed management: techniques,” *Gesunde Pflanzen*, vol. 60, no. 4, pp. 171–181, 2008.
- [46] R. Sharma, A. M. Hasan, J. Su, and M. Bhuiyan, “Improving weeds identification with a repository of agricultural pre-trained deep neural networks,” in *2021 6th International Conference on Innovative Technology in Intelligent System and Industrial Applications (CITISIA)*. IEEE, 2021, pp. 1–9.
- [47] A. Benelli, C. Cevoli, and A. Fabbri, “In-field hyperspectral imaging: An overview on the ground-based applications in agriculture,” *Journal of Agricultural Engineering*, vol. 51, no. 3, pp. 129–139, 2020.
- [48] W.-H. Su, “Advanced machine learning in point spectroscopy, rgb-and hyperspectral-imaging for automatic discriminations of crops and weeds: A review,” *Smart Cities*, vol. 3, no. 3, pp. 767–792, 2020.

- [49] M. R. Ahmed, B. G. Ram, C. Koparan, K. Howatt, Y. Zhang, and X. Sun, “Multiclass classification on soybean and weed species using a novel customized greenhouse robotic and hyperspectral combination system,” *Available at SSRN 4044574*.
- [50] A. Nasiri, M. Omid, A. Taheri-Garavand, and A. Jafari, “Deep learning-based precision agriculture through weed recognition in sugar beet fields,” *Sustainable Computing: Informatics and Systems*, p. 100759, 2022.
- [51] A. Subeesh, S. Bhole, K. Singh, N. Chandel, Y. Rajwade, K. Rao, S. Kumar, and D. Jat, “Deep convolutional neural network models for weed detection in polyhouse grown bell peppers,” *Artificial Intelligence in Agriculture*, vol. 6, pp. 47–54, 2022.
- [52] A. Hanson, *Computer vision systems*. Elsevier, 1978.
- [53] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.
- [54] D. Forsyth and J. Ponce, *Computer vision: A modern approach*. Prentice hall, 2011.
- [55] K. Agarwal and T. Singh, “Classification of skin cancer images using convolutional neural networks,” *arXiv preprint arXiv:2202.00678*, 2022.
- [56] A. Singla, L. Yuan, and T. Ebrahimi, “Food/non-food image classification and food categorization using pre-trained googlenet model,” in *Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management*, 2016, pp. 3–11.
- [57] M. T. Islam, B. N. K. Siddique, S. Rahman, and T. Jabid, “Food image classification with convolutional neural network,” in *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, vol. 3. IEEE, 2018, pp. 257–262.
- [58] S. Phiphiphatphaisit and O. Surinta, “Food image classification with improved mobilenet architecture and data augmentation,” in *Proceedings of the 2020 The 3rd International Conference on Information Science and System*, 2020, pp. 51–56.
- [59] T. Oduru, A. Jordan, and A. Park, “Healthy vs. unhealthy food images: Image classification of twitter images,” *International Journal of Environmental Research and Public Health*, vol. 19, no. 2, p. 923, 2022.
- [60] H. Ouchra and A. Belangour, “Satellite image classification methods and techniques: A survey,” in *2021 IEEE International Conference on Imaging Systems and Techniques (IST)*. IEEE, 2021, pp. 1–6.
- [61] W. Ramirez, P. Achancaray, and M. A. Pacheco, “A comparative study of deep learning architectures for classification of natural and human-made sea events in sar images,” *Discover Artificial Intelligence*, vol. 2, no. 1, pp. 1–13, 2022.
- [62] E. K. Hassan, H. M. Saeed, and A. H. T. Al-Ghrairi, “Classification and measurement of land cover of wildfires in australia using remote sensing,” *Iraqi Journal of Science*, pp. 420–430, 2022.
- [63] H. S. Abdullahi, R. Sheriff, and F. Mahieddine, “Convolution neural network in precision agriculture for plant image recognition and classification,” in *2017 Seventh International Conference on Innovative Computing Technology (INTECH)*, vol. 10. Ieee, 2017, pp. 256–272.
- [64] L. Hashemi-Beni and A. Gebrehiwot, “Deep learning for remote sensing image classification for agriculture applications,” *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 51–54, 2020.
- [65] I. M. Nasir, A. Bibi, J. H. Shah, M. A. Khan, M. Sharif, K. Iqbal, Y. Nam, and S. Kadry, “Deep learning-based classification of fruit diseases: An application for precision agriculture,” 2021.

- [66] A. S. Paymode and V. B. Malode, "Transfer learning for multi-crop leaf disease image classification using convolutional neural networks vgg," *Artificial Intelligence in Agriculture*, 2022.
- [67] Y. Wei, W. Xia, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan, "Cnn: Single-label to multi-label," *arXiv preprint arXiv:1406.5726*, 2014.
- [68] S. A. Shahriyar, K. M. R. Alam, S. S. Roy, and Y. Morimoto, "An approach for multi label image classification using single label convolutional neural network," in *2018 21st international conference of computer and information technology (ICCIT)*. IEEE, 2018, pp. 1–6.
- [69] E. Cakir, T. Heittola, H. Huttunen, and T. Virtanen, "Multi-label vs. combined single-label sound event detection with deep neural networks," in *2015 23rd European signal processing conference (EUSIPCO)*. IEEE, 2015, pp. 2551–2555.
- [70] "Difference between Multi-Class and Multi-Label Classification", Analytics Vidhya, Jul. 20, 2021. <https://www.analyticsvidhya.com/blog/2021/07/demystifying-the-difference-between-multi-class-and-multi-label-classification-problem-statements-in-deep-learning/> [accessed Feb. 07, 2022].
- [71] "Main Challenges in Image Classification | by Rafay Gilani | Towards Data Science". <https://towardsdatascience.com/main-challenges-in-image-classification-ba24dc78b558> [accessed Feb. 07, 2022].
- [72] J. Zhang, Y. Xie, Q. Wu, and Y. Xia, "Medical image classification using synergic deep learning," *Medical image analysis*, vol. 54, pp. 10–19, 2019.
- [73] T. Tai and M. Toda, "Adapting intra-class variations for sar image classification," in *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2021, pp. 2653–2657.
- [74] M. Koc, S. Ergin, M. B. Gulmezoglu, M. Fidan, O. N. Gerek, and A. Barkana, "Necessary conditions for successful application of intra-and inter-class common vector classifiers," *Arabian Journal for Science and Engineering*, pp. 1–13, 2022.
- [75] S. Ray, "A quick review of machine learning algorithms," in *2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE, 2019, pp. 35–39.
- [76] K. Sanghvi, A. Aralkar, S. Sanghvi, and I. Saha, "A survey on image classification techniques," *Available at SSRN 3754116*, 2020.
- [77] H. Singh, V. Sharma, and D. Singh, "Comparative analysis of proficiencies of various textures and geometric features in breast mass classification using k-nearest neighbor," *Visual Computing for Industry, Biomedicine, and Art*, vol. 5, no. 1, pp. 1–19, 2022.
- [78] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees," 1984.
- [79] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 2017.
- [80] I. Povkhan and M. Lupei, "The algorithmic classification trees," in *2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)*. IEEE, 2020, pp. 37–43.
- [81] Z.-H. Zhou and Z.-Q. Chen, "Hybrid decision tree," *Knowledge-based systems*, vol. 15, no. 8, pp. 515–528, 2002.
- [82] A. Shehadeh, O. Alshboul, R. E. Al Mamlook, and O. Hamedat, "Machine learning models for predicting the residual value of heavy construction equipment: An evaluation of modified decision tree, lightgbm, and xgboost regression," *Automation in Construction*, vol. 129, p. 103827, 2021.

- [83] W. Deelder, G. Napier, S. Campino, L. Palla, J. Phelan, and T. G. Clark, “A modified decision tree approach to improve the prediction and mutation discovery for drug resistance in mycobacterium tuberculosis,” *BMC genomics*, vol. 23, no. 1, pp. 1–7, 2022.
- [84] N. Horning *et al.*, “Random forests: An algorithm for image classification and generation of continuous fields data sets,” in *Proceedings of the International Conference on Geoinformatics for Spatial Infrastructure Development in Earth and Allied Sciences, Osaka, Japan*, vol. 911, 2010.
- [85] C. Min and Z. Haijiang, “Research on application of improved random forest in medical ultrasound image classification,” in *Journal of Physics: Conference Series*, vol. 1584, no. 1. IOP Publishing, 2020, p. 012007.
- [86] A. Zafari, R. Zurita-Milla, and E. Izquierdo-Verdiguier, “A multiscale random forest kernel for land cover classification,” *IEEE Journal of selected topics in applied earth observations and remote sensing*, vol. 13, pp. 2842–2852, 2020.
- [87] P. Mekha and N. Teeyasuksaet, “Image classification of rice leaf diseases using random forest algorithm,” in *2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering*. IEEE, 2021, pp. 165–169.
- [88] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [89] X. Sun, L. Liu, H. Wang, W. Song, and J. Lu, “Image classification via support vector machine,” in *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, vol. 1. IEEE, 2015, pp. 485–489.
- [90] W. H. Land and J. D. Schaffer, “The support vector machine,” in *The art and science of machine intelligence*. Springer, 2020, pp. 45–76.
- [91] D. A. Pisner and D. M. Schnyer, “Support vector machine,” in *Machine learning*. Elsevier, 2020, pp. 101–121.
- [92] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, “A comprehensive survey on support vector machine classification: Applications, challenges and trends,” *Neurocomputing*, vol. 408, pp. 189–215, 2020.
- [93] H. Kong *et al.*, “Research on classification of high spatial resolution remote sensing image based on svm,” *Geoscience and Remote Sensing*, vol. 4, no. 1, pp. 1–6, 2021.
- [94] G. M. Foody and A. Mathur, “A relative evaluation of multiclass image classification by support vector machines,” *IEEE Transactions on geoscience and remote sensing*, vol. 42, no. 6, pp. 1335–1343, 2004.
- [95] M. Lapin, M. Hein, and B. Schiele, “Top-k multiclass svm,” *Advances in Neural Information Processing Systems*, vol. 28, 2015.
- [96] X. Li, “Chinese language and literature online resource classification algorithm based on improved svm,” *Scientific Programming*, vol. 2022, 2022.
- [97] L. P. Osco, J. M. Junior, A. P. M. Ramos, L. A. de Castro Jorge, S. N. Fatholahi, J. de Andrade Silva, E. T. Matsubara, H. Pistori, W. N. Gonçalves, and J. Li, “A review on deep learning in uav remote sensing,” *International Journal of Applied Earth Observation and Geoinformation*, vol. 102, p. 102456, 2021.
- [98] M. A. Abdou, “Literature review: efficient deep neural networks techniques for medical image analysis,” *Neural Computing and Applications*, pp. 1–22, 2022.

- [99] D. Wang, W. Cao, F. Zhang, Z. Li, S. Xu, and X. Wu, "A review of deep learning in multiscale agricultural sensing," *Remote Sensing*, vol. 14, no. 3, p. 559, 2022.
- [100] K. B. Obaid, S. Zeebaree, O. M. Ahmed *et al.*, "Deep learning models based on image classification: a review," *International Journal of Science and Business*, vol. 4, no. 11, pp. 75–81, 2020.
- [101] B. Jena, S. Saxena, G. K. Nayak, L. Saba, N. Sharma, and J. S. Suri, "Artificial intelligence-based hybrid deep learning models for image classification: The first narrative review," *Computers in Biology and Medicine*, vol. 137, p. 104803, 2021.
- [102] D. Roy and K. Murari, "Image classification through deep neural learning: A review."
- [103] M. Shadman Roodposhti, J. Aryal, A. Lucieer, and B. A. Bryan, "Uncertainty assessment of hyperspectral image classification: Deep learning vs. random forest," *Entropy*, vol. 21, no. 1, p. 78, 2019.
- [104] L. Medsker and L. C. Jain, *Recurrent neural networks: design and applications*. CRC press, 1999.
- [105] A. Aggarwal, M. Mittal, and G. Battineni, "Generative adversarial network: An overview of theory and applications," *International Journal of Information Management Data Insights*, vol. 1, no. 1, p. 100004, 2021.
- [106] A. Decelle and C. Furtlehner, "Restricted boltzmann machine: Recent advances and mean-field theory," *Chinese Physics B*, vol. 30, no. 4, p. 040202, 2021.
- [107] T. Sarvini, T. Sneha, S. G. GS, S. Sushmitha, and R. Kumaraswamy, "Performance comparison of weed detection algorithms," in *2019 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2019, pp. 0843–0847.
- [108] B. Liu and R. Bruch, "Weed detection for selective spraying: a review," *Current Robotics Reports*, vol. 1, no. 1, pp. 19–26, 2020.
- [109] M. Renukadvi and C. M. Sulaikha, "A survey on image processing methodologies for crop and weed detection," *Annals of the Romanian Society for Cell Biology*, vol. 25, no. 6, pp. 10 251–10 260, 2021.
- [110] A. M. Mishra and V. Gautam, "Weed species identification in different crops using precision weed management: A review," in *Proc. of CEUR Workshop*, 2021, pp. 180–194.
- [111] Z. Wu, Y. Chen, B. Zhao, X. Kang, and Y. Ding, "Review of weed detection methods based on computer vision," *Sensors*, vol. 21, no. 11, p. 3647, 2021.
- [112] P. Saini, "Recent advancement of weed detection in crops using artificial intelligence and deep learning: A review," *Advances in Energy Technology*, pp. 631–640, 2022.
- [113] M. Weis, T. Rumpf, R. Gerhards, and L. Plümer, "Comparison of different classification algorithms for weed detection from images based on shape parameters," *Bornimer Agrartechn. Ber*, vol. 69, pp. 53–64, 2009.
- [114] C. A. P. Rojas, L. E. S. Guzman, and N. F. V. Toledo, "A comparative analysis of weed images classification approaches in vegetables crops," *Engineering Journal*, vol. 21, no. 2, pp. 81–98, 2017.
- [115] N. Islam, M. M. Rashid, S. Wibowo, C.-Y. Xu, A. Morshed, S. A. Wasimi, S. Moore, and S. M. Rahman, "Early weed detection using image processing and machine learning techniques in an australian chilli farm," *Agriculture*, vol. 11, no. 5, p. 387, 2021.
- [116] D. Rainville, A. Durand, F.-A. Fortin, K. Tanguy, X. Maldague, B. Panneton, M.-J. Simard *et al.*, "Bayesian classification and unsupervised learning for isolating weeds in row crops," *Pattern Analysis and Applications*, vol. 17, no. 2, pp. 401–414, 2014.

- [117] M. Mursalin and M. Mesbah-Ul-Awal, "Towards classification of weeds through digital image," in *2014 Fourth International Conference on Advanced Computing & Communication Technologies*. IEEE, 2014, pp. 1-4.
- [118] I. D. García-Santillán and G. Pajares, "On-line crop/weed discrimination through the mahalanobis distance from images in maize fields," *Biosystems engineering*, vol. 166, pp. 28-43, 2018.
- [119] S. K. Mathanker, P. R. Weckler, R. K. Taylor, and G. Fan, "Adaboost and support vector machine classifiers for automatic weed control: Canola and wheat," in *2010 Pittsburgh, Pennsylvania, June 20-June 23, 2010*. American Society of Agricultural and Biological Engineers, 2010, p. 1.
- [120] J. Ahmad, K. Muhammad, I. Ahmad, W. Ahmad, M. L. Smith, L. N. Smith, D. K. Jain, H. Wang, and I. Mehmood, "Visual features based boosted classification of weeds for real-time selective herbicide sprayer systems," *Computers in Industry*, vol. 98, pp. 23-33, 2018.
- [121] G. Khurana and N. K. Bawa, "Performance analysis of k-nearest neighbor method for the weed detection."
- [122] ——, "Weed detection approach using feature extraction and knn classification," in *Advances in Electro-mechanical Technologies*. Springer, 2021, pp. 671-679.
- [123] Y. Guo, C. Du, Y. Zhao, T.-F. Ting, and T. A. Rothfus, "Two-level k-nearest neighbors approach for invasive plants detection and classification," *Applied Soft Computing*, vol. 108, p. 107523, 2021.
- [124] A. A. J. Sinlae, D. Alamsyah, L. Suhery, and F. Fatmayati, "Classification of broadleaf weeds using a combination of k-nearest neighbor (knn) and principal component analysis (pca)," *Sinkron: jurnal dan penelitian teknik informatika*, vol. 7, no. 1, pp. 93-100, 2022.
- [125] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (pca)," *Computers & Geosciences*, vol. 19, no. 3, pp. 303-342, 1993.
- [126] R. Sohail, Q. Nawaz, I. Hamid, H. Amin, J. N. Chaudhary, S. M. M. Gilani, and I. Mumtaz, "A novel machine learning based algorithm to detect weeds in soybean crop," *Pak. J. Agri. Sci*, vol. 58, no. 3, pp. 1007-1015, 2021.
- [127] A. I. De Castro, J. Torres-Sánchez, J. M. Peña, F. M. Jiménez-Brenes, O. Csillik, and F. López-Granados, "An automatic random forest-obia algorithm for early weed mapping between and within crop rows using uav imagery," *Remote Sensing*, vol. 10, no. 2, p. 285, 2018.
- [128] Z. H. Kok, A. R. M. Shariff, M. S. M. Alfatni, and S. Khairunniza-Bejo, "Support vector machine in precision agriculture: A review," *Computers and Electronics in Agriculture*, vol. 191, p. 106546, 2021.
- [129] J. Brahim, R. Loubna, and F. Noureddine, "Rnn-and cnn-based weed detection for crop improvement: An overview," *Foods and Raw materials*, vol. 9, no. 2, pp. 387-396, 2021.
- [130] L. Hashemi-Beni, A. Gebrehiwot, A. Karimoddini, A. Shahbazi, and F. Dorbu, "Deep convolutional neural networks for weeds and crops discrimination from uas imagery," *Front. Remote Sens. 3: 755939. doi: 10.3389/frsen*, 2022.
- [131] Y. Li, M. Al-Sarayreh, K. Irie, D. Hackell, G. Bourdot, M. M. Reis, and K. Ghamkhar, "Identification of weeds based on hyperspectral imaging and machine learning," *Frontiers in Plant Science*, vol. 11, p. 2324, 2021.
- [132] A. J. Ishak, M. M. Mustafa, N. M. Tahir, and A. Hussain, "Weed detection system using support vector machine," in *2008 International Symposium on Information Theory and Its Applications*. IEEE, 2008, pp. 1-4.

- [133] A. Tellaeché, G. Pajares, X. P. Burgos-Artizzu, and A. Ribeiro, “A computer vision approach for weeds identification through support vector machines,” *Applied Soft Computing*, vol. 11, no. 1, pp. 908–915, 2011.
- [134] Y. Chen, Z. Wu, B. Zhao, C. Fan, and S. Shi, “Weed and corn seedling detection in field based on multi feature fusion and support vector machine,” *Sensors*, vol. 21, no. 1, p. 212, 2021.
- [135] S. Rani, P. S. Kumar, R. Priyadharsini, S. J. Srividya, and S. Harshana, “Automated weed detection system in smart farming for developing sustainable agriculture,” *International Journal of Environmental Science and Technology*, pp. 1–12, 2021.
- [136] S. Soni, “Support vector machine and k-means algorithm for weed classification in the crops,” *Advances in Computer Science and Information Technology*, p. 37.
- [137] A. Ibrahim, F. Anayi, M. Packianather, and O. Al-Omari, “Novel hybrid invasive weed optimization and machine learning approach for fault detection,” in *2021 56th International Universities Power Engineering Conference (UPEC)*. IEEE, 2021, pp. 1–6.
- [138] S. Veeragandham and H. Santhi, “A detailed review on challenges and imperatives of various cnn algorithms in weed detection,” in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*. IEEE, 2021, pp. 1068–1073.
- [139] A. Tannouche, A. Gaga, M. Boutalline, and S. Belhouideg, “Weeds detection efficiency through different convolutional neural networks technology,” *International Journal of Electrical and Computer Engineering*, vol. 12, no. 1, p. 1048, 2022.
- [140] J. Zhuang, X. Li, M. Bagavathiannan, X. Jin, J. Yang, W. Meng, T. Li, L. Li, Y. Wang, Y. Chen *et al.*, “Evaluation of different deep convolutional neural networks for detection of broadleaf weed seedlings in wheat,” *Pest Management Science*, 2022.
- [141] S. A. Siddiqui, N. Fatima, and A. Ahmad, “Neural network based smart weed detection system,” in *2021 International Conference on Communication, Control and Information Sciences (ICCISc)*, vol. 1. IEEE, 2021, pp. 1–5.
- [142] M. Dou, Z. Hong, and M. Shi, “An improved efficient convolutional neural network for weed seedlings detection,” in *2021 International Conference on Culture-oriented Science & Technology (ICCST)*. IEEE, 2021, pp. 285–289.
- [143] C. Hu, B. B. Sapkota, J. A. Thomasson, and M. V. Bagavathiannan, “Influence of image quality and light consistency on the performance of convolutional neural networks for weed mapping,” *Remote Sensing*, vol. 13, no. 11, p. 2140, 2021.
- [144] B. Espejo-Garcia, N. Mylonas, L. Athanasakos, S. Fountas, and I. Vasilakoglou, “Towards weeds identification assistance through transfer learning,” *Computers and Electronics in Agriculture*, vol. 171, p. 105306, 2020.
- [145] M. Ofori and O. El-Gayar, “An approach for weed detection using cnns and transfer learning,” 2021.
- [146] B. Espejo-Garcia, N. Mylonas, L. Athanasakos, E. Vali, and S. Fountas, “Combining generative adversarial networks and agricultural transfer learning for weeds identification,” *Biosystems Engineering*, vol. 204, pp. 79–89, 2021.
- [147] D. Chen, Y. Lu, Z. Li, and S. Young, “Performance evaluation of deep transfer learning on multiclass identification of common weed species in cotton production systems,” *arXiv preprint arXiv:2110.04960*, 2021.

- [148] A. Farooq, X. Jia, J. Hu, and J. Zhou, “Transferable convolutional neural network for weed mapping with multisensor imagery,” *IEEE Transactions on Geoscience and Remote Sensing*, 2021.
- [149] S. Mirjalili, “Genetic algorithm,” in *Evolutionary algorithms and neural networks*. Springer, 2019, pp. 43–55.
- [150] A. Lambora, K. Gupta, and K. Chopra, “Genetic algorithm-a literature review,” in *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE, 2019, pp. 380–384.
- [151] A. N. Sloss and S. Gustafson, “2019 evolutionary algorithms review,” *Genetic programming Theory and practice XVII*, pp. 307–344, 2020.
- [152] S. Katoch, S. S. Chauhan, and V. Kumar, “A review on genetic algorithm: past, present, and future,” *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, 2021.
- [153] Y. Bi, B. Xue, and M. Zhang, “Evolutionary computation and genetic programming,” in *Genetic Programming for Image Classification*. Springer, 2021, pp. 49–74.
- [154] H. Tamaki, H. Kita, and S. Kobayashi, “Multi-objective optimization by genetic algorithms: A review,” in *Proceedings of IEEE international conference on evolutionary computation*. IEEE, 1996, pp. 517–522.
- [155] S. Sivanandam and S. Deepa, “Genetic algorithm optimization problems,” in *Introduction to genetic algorithms*. Springer, 2008, pp. 165–209.
- [156] H. A. Daham and H. J. Mohammed, “An evolutionary algorithm approach for vehicle routing problems with backhauls,” *Materials Today: Proceedings*, 2021.
- [157] A. Slowik and H. Kwasnicka, “Evolutionary algorithms and their applications to engineering problems,” *Neural Computing and Applications*, vol. 32, no. 16, pp. 12 363–12 379, 2020.
- [158] C.-C. Lai and C.-Y. Chang, “A hierarchical evolutionary algorithm for automatic medical image segmentation,” *Expert Systems with Applications*, vol. 36, no. 1, pp. 248–259, 2009.
- [159] M. Jabbar, B. Deekshatulu, and P. Chandra, “An evolutionary algorithm for heart disease prediction,” in *International Conference on Information Processing*. Springer, 2012, pp. 378–389.
- [160] J. Biethahn and V. Nissen, *Evolutionary algorithms in management applications*. Springer Science & Business Media, 2012.
- [161] D. Agnelli, A. Bollini, and L. Lombardi, “Image classification: an evolutionary approach,” *Pattern Recognition Letters*, vol. 23, no. 1-3, pp. 303–309, 2002.
- [162] C.-W. Bong and M. Rajeswari, “Multi-objective nature-inspired clustering and classification techniques for image segmentation,” *Applied soft computing*, vol. 11, no. 4, pp. 3271–3282, 2011.
- [163] Y. Bi, B. Xue, and M. Zhang, “An automated ensemble learning framework using genetic programming for image classification,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 365–373.
- [164] A. Zafra, “Ying bi, bing xue, mengjie zhang: Genetic programming for image classification—an automated approach to feature learning,” *Genetic Programming and Evolvable Machines*, pp. 1–2, 2022.
- [165] N. M. Aszemi and P. Dominic, “Hyperparameter optimization in convolutional neural network using genetic algorithms,” *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 6, 2019.

- [166] X. Xiao, M. Yan, S. Basodi, C. Ji, and Y. Pan, “Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm,” *arXiv preprint arXiv:2006.12703*, 2020.
- [167] R. Kumar Pandey, A. Kumar, A. Mandal, and B. Vaferi, “Genetic algorithm optimization of deep structured classifier-predictor models for pressure transient analysis,” *Journal of Energy Resources Technology*, pp. 1–22, 2022.
- [168] Y. Bi, B. Xue, and M. Zhang, “An automatic feature extraction approach to image classification using genetic programming,” in *International Conference on the Applications of Evolutionary Computation*. Springer, 2018, pp. 421–438.
- [169] K. Senthil Kumar, K. Venkatalakshmi, and K. Karthikeyan, “Lung cancer detection using image segmentation by means of various evolutionary algorithms,” *Computational and mathematical methods in medicine*, vol. 2019, 2019.
- [170] N. G. RoopaKumari and N. Kumar, “Image segmentation using improved genetic algorithm,” *Int. J. of Engineering and Advanced Technology (IJEAT)*, vol. 9, no. 1, 2019.
- [171] L. Khrissi, N. El Akkad, H. Satori, and K. Satori, “Image segmentation based on k-means and genetic algorithms,” in *Embedded Systems and Artificial Intelligence*. Springer, 2020, pp. 489–497.
- [172] M. Iqbal, M. Zhang, and B. Xue, “Improving classification on images by extracting and transferring knowledge in genetic programming,” in *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2016, pp. 3582–3589.
- [173] A. Peerlinck, J. Sheppard, J. Pastorino, and B. Maxwell, “Optimal design of experiments for precision agriculture using a genetic algorithm,” in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 1838–1845.
- [174] B. A. Lease, “Weed/plant classification using evolutionary optimised ensemble based on local binary patterns,” Ph.D. dissertation, Curtin University, 2022.
- [175] L. Tang, L. Tian, and B. L. Steward, “Color image segmentation with genetic algorithm for in-field weed sensing,” *Transactions of the ASAE*, vol. 43, no. 4, p. 1019, 2000.
- [176] R. R. d. Silva, M. C. Escarpinati, and A. R. Backes, “Sugarcane crop line detection from uav images using genetic algorithm and radon transform,” *Signal, Image and Video Processing*, vol. 15, no. 8, pp. 1723–1730, 2021.
- [177] H. Al-Sahaf, Y. Bi, Q. Chen, A. Lensen, Y. Mei, Y. Sun, B. Tran, B. Xue, and M. Zhang, “A survey on evolutionary machine learning,” *Journal of the Royal Society of New Zealand*, vol. 49, no. 2, pp. 205–228, 2019.
- [178] H. T. Ünal and F. Başçiftçi, “Evolutionary design of neural network architectures: a review of three decades of research,” *Artificial Intelligence Review*, pp. 1–80, 2021.
- [179] P. Darnowsky, “Image classification with evolved convolutional neural networks,” Ph.D. dissertation, 2022.
- [180] Y. Bi, B. Xue, and M. Zhang, “An evolutionary deep learning approach using genetic programming with convolution operators for image classification,” in *2019 IEEE congress on evolutionary computation (CEC)*. IEEE, 2019, pp. 3197–3204.
- [181] M. Suganuma, S. Shirakawa, and T. Nagao, “A genetic programming approach to designing convolutional neural network architectures,” in *Proceedings of the genetic and evolutionary computation conference*, 2017, pp. 497–504.

- [182] B. Evans, H. Al-Sahaf, B. Xue, and M. Zhang, “Evolutionary deep learning: A genetic programming approach to image classification,” in *2018 IEEE congress on evolutionary computation (CEC)*. IEEE, 2018, pp. 1–6.
- [183] Y. Bi, B. Xue, and M. Zhang, *Genetic Programming for Image Classification: An Automated Approach to Feature Learning*. Springer Nature, 2021, vol. 24.
- [184] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, “Evolving deep convolutional neural networks for image classification,” *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, 2019.
- [185] T. Hassanzadeh, D. Essam, and R. Sarker, “Evodcnn: An evolutionary deep convolutional neural network for image classification,” *Neurocomputing*, 2022.
- [186] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain, and A. J. Aljaaf, “A systematic review on supervised and unsupervised machine learning algorithms for data science,” *Supervised and unsupervised learning for data science*, pp. 3–21, 2020.
- [187] K. K. Hiran, R. K. Jain, K. Lakhwani, and R. Doshi, *Machine Learning: Master Supervised and Unsupervised Learning Algorithms with Real Examples (English Edition)*. BPB Publications, 2021.
- [188] K. Das and R. N. Behera, “A survey on machine learning: concept, algorithms and applications,” *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 5, no. 2, pp. 1301–1309, 2017.
- [189] C. Janiesch, P. Zschech, and K. Heinrich, “Machine learning and deep learning,” *Electronic Markets*, vol. 31, no. 3, pp. 685–695, 2021.
- [190] I. H. Sarker, “Machine learning: Algorithms, real-world applications and research directions,” *SN Computer Science*, vol. 2, no. 3, pp. 1–21, 2021.
- [191] A. M. Deiana, N. Tran, J. Agar, M. Blott, G. Di Guglielmo, J. Duarte, P. Harris, S. Hauck, M. Liu, M. S. Neubauer *et al.*, “Applications and techniques for fast machine learning in science,” *Frontiers in big Data*, vol. 5, 2022.
- [192] J. Zupan, “Introduction to artificial neural network (ann) methods: what they are and how to use them,” *Acta Chimica Slovenica*, vol. 41, pp. 327–327, 1994.
- [193] H. Kukreja, N. Bharath, C. Siddesh, and S. Kuldeep, “An introduction to artificial neural network,” *Int J Adv Res Innov Ideas Educ*, vol. 1, pp. 27–30, 2016.
- [194] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological review*, vol. 65, no. 6, p. 386, 1958.
- [195] J. Swan, E. Nivel, N. Kant, J. Hedges, T. Atkinson, and B. Steunebrink, “Challenges for deep learning,” in *The Road to General Intelligence*. Springer, 2022, pp. 23–32.
- [196] K. Singh, D. Singh, and N. Mishra, “Convolutional neural networks and its architecture,” *architecture*, vol. 6, no. S1, pp. 9183–9190.
- [197] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, “Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions,” *Journal of big Data*, vol. 8, no. 1, pp. 1–74, 2021.
- [198] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, “Understanding batch normalization,” *Advances in neural information processing systems*, vol. 31, 2018.
- [199] M. Awais, M. T. B. Iqbal, and S.-H. Bae, “Revisiting internal covariate shift for batch normalization,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 11, pp. 5082–5092, 2020.

- [200] "ImageNet Large Scale Visual Recognition Challenge "(ILSVRC), ImageNet, Jul. 20, 2021. <https://www.image-net.org/challenges/LSVRC/> [accessed Feb. 07, 2022].
- [201] "26TH International Conference on Pattern Recognition"(ILSVRC), Jul. 20, 2021. <https://www.icpr2022.com/call-for-challenges/> [accessed Jul. 07, 2022].
- [202] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [203] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1-9.
- [204] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.
- [205] D. M. Kline and V. L. Berardi, "Revisiting squared-error and cross-entropy functions for training neural network classifiers," *Neural Computing & Applications*, vol. 14, no. 4, pp. 310-318, 2005.
- [206] R. Rojas, "The backpropagation algorithm," in *Neural networks*. Springer, 1996, pp. 149-182.
- [207] M. G. Abdolrasol, S. S. Hussain, T. S. Ustun, M. R. Sarker, M. A. Hannan, R. Mohamed, J. A. Ali, S. Mekhilef, and A. Milad, "Artificial neural networks based optimization techniques: A review," *Electronics*, vol. 10, no. 21, p. 2689, 2021.
- [208] X. Ying, "An overview of overfitting and its solutions," in *Journal of physics: Conference series*, vol. 1168, no. 2. IOP Publishing, 2019, p. 022022.
- [209] R. Dwivedi, C. Singh, B. Yu, and M. J. Wainwright, "Revisiting complexity and the bias-variance tradeoff," *arXiv preprint arXiv:2006.10189*, 2020.
- [210] Z. Yang, Y. Yu, C. You, J. Steinhardt, and Y. Ma, "Rethinking bias-variance trade-off for generalization of neural networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 10767-10777.
- [211] Y. Tian and Y. Zhang, "A comprehensive survey on regularization strategies in machine learning," *Information Fusion*, vol. 80, pp. 146-166, 2022.
- [212] E. WOLP and W. MACREADY, "No free lunch theorems for search," SH-TR-95-02-010Santa Fe Institute. Santa Fe. New Mexico, Tech. Rep., 1995.
- [213] D. B. Fogel and L. J. Fogel, "An introduction to evolutionary programming," in *European conference on artificial evolution*. Springer, 1995, pp. 21-33.
- [214] A. Huning, "Evolutionsstrategie. optimierung technischer systeme nach prinzipien der biologischen evolution," 1976.
- [215] R. Storn and K. Price, "Differential evolution a simple evolution strategy for fast optimization," *Dr. Dobb's Journal*, vol. 22, no. 4, pp. 18-24, 1997.
- [216] J. R. Koza *et al.*, *Genetic programming II*. MIT press Cambridge, 1994, vol. 17.
- [217] R. M. Karp, "An introduction to randomized algorithms," *Discrete Applied Mathematics*, vol. 34, no. 1-3, pp. 165-201, 1991.
- [218] J. Hesser and R. Männer, "Towards an optimal mutation probability for genetic algorithms," in *International Conference on Parallel Problem Solving from Nature*. Springer, 1990, pp. 23-32.
- [219] V. Patil and D. Pawar, "The optimal crossover or mutation rates in genetic algorithm: a review," *International Journal of Applied Engineering and Technology*, vol. 5, no. 3, pp. 38-41, 2015.

- [220] K. A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems*. University of Michigan, 1975.
- [221] J. Knowles, D. Corne, and K. Deb, *Multiobjective problem solving from nature: from concepts to applications*. Springer Science & Business Media, 2007.
- [222] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, “Multiobjective evolutionary algorithms: A survey of the state of the art,” *Swarm and evolutionary computation*, vol. 1, no. 1, pp. 32–49, 2011.
- [223] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, “A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii,” in *International conference on parallel problem solving from nature*. Springer, 2000, pp. 849–858.
- [224] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.
- [225] N. Srinivas and K. Deb, “Muultiobjective optimization using nondominated sorting in genetic algorithms,” *Evolutionary computation*, vol. 2, no. 3, pp. 221–248, 1994.
- [226] *Sistema de colas SLURM*. CSIC 2022 [online] Available at: <http://sitios.csic.es/web/calcuocientífico/sistema-de-colas-slurm> [accessed Feb. 22, 2022].
- [227] SLURM workload Version 22.05. 2022 [online] Available at: <https://slurm.schedmd.com/overview.html> [accessed Feb. 18, 2022].
- [228] T. Ylonen, “Ssh–secure login connections over the internet,” in *Proceedings of the 6th USENIX Security Symposium*, vol. 37, 1996, pp. 40–52.
- [229] T. Ylonen and C. Lonvick, “The secure shell (ssh) connection protocol,” Tech. Rep., 2006.
- [230] Cesga - Centro de Supercomputación de Galicia. 2022. *FinisTerrae III User Guide - Basic Comands*. [online] Available at: [https://cesga-docs.gitlab.io/ft3-user-guide/batch\\_basic\\_commands.html](https://cesga-docs.gitlab.io/ft3-user-guide/batch_basic_commands.html) [Accessed 7 June 2022].
- [231] Linux.org [online] Available at: <https://www.linux.org> [accessed Jun. 20, 2022].
- [232] *MacOS Monterey*. Apple [online] Available at: <https://www.apple.com/es/macos/monterey/> [accessed Jun. 20, 2022].
- [233] Overleaf [online] Available at: <https://es.overleaf.com> [accessed Jun. 20, 2022].
- [234] Anaconda [online] Available at: <https://www.anaconda.com> [accessed Jun. 20, 2022].
- [235] Conda. Miniconda [online] Available at: <https://docs.conda.io/en/latest/miniconda.html> [accessed Jun. 20, 2022].
- [236] Python [online] Available at: <https://www.python.org/downloads/> [accessed Jun. 20, 2022].
- [237] Tensorflow [online] Available at: <https://www.tensorflow.org> [accessed Jun. 20, 2022].
- [238] Keras [online] Available at: <https://keras.io> [accessed Jun. 20, 2022].
- [239] CUDA [online] Available at: <https://developer.nvidia.com/cuda-downloads> [accessed Jun. 20, 2022].
- [240] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, “DEAP: Evolutionary algorithms made easy,” *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.
- [241] Tensorflow. 2022. *Image Classification, dataset catalog - “deep\_weeds”*. [online] Available at: [https://www.tensorflow.org/datasets/catalog/deep\\_weeds](https://www.tensorflow.org/datasets/catalog/deep_weeds) [accessed Feb. 20, 2022].

- [242] Python Package Index. 2022 - “*tfds-nightly*”. [online] Available at: <https://pypi.org/project/tfds-nightly/> [accessed Feb. 20, 2022].
- [243] Scikit Learn, Machine Learning in Python. 2022 - *sklearn.model\_selection.train\_test\_split*. [online] Available at: [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.train\\_test\\_split.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) [accessed Feb. 26, 2022].
- [244] Pillow (PIL Fork), 2022 - *PIL, Python Imaging Library*. [online] Available at: <https://pillow.readthedocs.io/en/stable/> [accessed Feb. 26, 2022].
- [245] Tensorflow, 2022 - data, *Datasetfrom\_tensor\_slices*. [online] Available at: [https://www.tensorflow.org/api\\_docs/python/tf/data/Datasetfrom\\_tensor\\_slices](https://www.tensorflow.org/api_docs/python/tf/data/Datasetfrom_tensor_slices) [accessed Feb. 28, 2022].
- [246] *DEAP Documentation*. DEAP 2022 [online] Available at: <https://deap.readthedocs.io/en/master/> [accessed Feb. 28, 2022].
- [247] S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. R. Salakhutdinov, and Y. Bengio, “Architectural complexity measures of recurrent neural networks,” *Advances in neural information processing systems*, vol. 29, 2016.
- [248] R. Xin, J. Zhang, and Y. Shao, “Complex network classification with convolutional neural network,” *Tsinghua Science and technology*, vol. 25, no. 4, pp. 447–457, 2020.
- [249] J. Das, G. Cross, C. Qu, A. Makineni, P. Tokekar, Y. Mulgaonkar, and V. Kumar, “Devices, systems, and methods for automated monitoring enabling precision agriculture,” in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2015, pp. 462–469.
- [250] F. J. Ferrández-Pastor, J. M. García-Chamizo, M. Nieto-Hidalgo, and J. Mora-Martínez, “Precision agriculture design method using a distributed computing architecture on internet of things context,” *Sensors*, vol. 18, no. 6, p. 1731, 2018.
- [251] A. Danton, J.-C. Roux, B. Dance, C. Cariou, and R. Lenain, “Development of a spraying robot for precision agriculture: An edge following approach,” in *2020 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2020, pp. 267–272.
- [252] *Open CV python*, Python Package Index [online] Available at: <https://pypi.org/project/opencv-python/> [accessed Jun. 1, 2022].
- [253] *tf.distribute.MirroredStrategy*, Tensorflow [online] Available at: [https://www.tensorflow.org/api\\_docs/python/tf/distribute/MirroredStrategy](https://www.tensorflow.org/api_docs/python/tf/distribute/MirroredStrategy) [accessed Jun. 10, 2022].
- [254] X. Hu, L. Chu, J. Pei, W. Liu, and J. Bian, “Model complexity of deep learning: A survey,” *Knowledge and Information Systems*, vol. 63, no. 10, pp. 2585–2619, 2021.
- [255] *Models and Layers*, Tensorflow [online] Available at: [https://www.tensorflow.org/js/guide/models\\_and\\_layers](https://www.tensorflow.org/js/guide/models_and_layers) [accessed Jun. 20, 2022].
- [256] Scikit Learn, Machine Learning in Python. 2022 - *sklearn.linear\_model.LinearRegression*. [online] Available at: [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LinearRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html) [accessed Jun. 26, 2022].