UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA INFORMATICA

Algoritmos meméticos para reducir datos de entrenamiento en modelos de aprendizaje profundo convolucionales

---

Autor

José Ruiz López (alumno)

Directores

Daniel Molina Cabrera (tutor)

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Mayo de 2025

Algoritmos meméticos para reducir datos de entrenamiento en modelos de aprendizaje profundo convolucionales

José Ruiz López (alumno)

Palabras clave: Algoritmos meméticos, Imágenes, Modelos de Aprendizaje profundo convolucionales

Resumen

Los **modelos de Aprendizaje Profundo** (Deep Learning) han supuesto un verdadero hito en la **Inteligencia Artificial**, ya que son capaces de procesar grandes volúmenes de datos, además de reconocer patrones sumamente complejos. Dentro de estos, los **modelos convolucionales** se han destacado como particularmente efectivos a la hora de identificar objetos y características en imágenes —una capacidad esencial para muchas aplicaciones modernas—. Sin embargo, a diferencia de los seres humanos, estos modelos requieren una gran cantidad de datos de entrenamiento para cada categoría que deben aprender. Esto implica un proceso de entrenamiento más largo y, muchas veces, la recolección de los datos necesarios puede ser problemática, según el tipo de información que se necesite.

Además de la dificultad en la obtención de datos, las nuevas normativas europeas en torno a la inteligencia artificial establecen la necesidad de auditar no solo los modelos, sino también los datos utilizados para entrenarlos, especialmente cuando se trata de aplicaciones de IA que manejan datos sensibles. Estas auditorías, por su propia naturaleza, se volverán más complejas conforme aumente el tamaño del conjunto de entrenamiento. Por lo tanto, se vuelve completamente necesario desarrollar estrategias que permitan **reducir el tamaño de los conjuntos de datos de entrenamiento** intentado comprometer la calidad del modelo lo mínimo posible.

En este trabajo, proponemos el uso de **algoritmos meméticos** para establecer un proceso de reducción del conjunto de **entrenamiento**, lo que se conoce como **selección de instancias**. La idea es seleccionar un conjunto reducido de imágenes representativas que, junto con las técnicas de aumento de datos, sean suficientes para entrenar modelos convolucionales de manera óptima. De este modo, se podría reducir significativamente el tamaño del conjunto de entrenamiento, manteniendo la calidad del aprendizaje y, a su vez, facilitando tanto el proceso de auditoría como la eficiencia computacional del sistema.

Memetic Algorithms for Reducing Training Data in Convolutional Deep Learning Models

José, Ruiz López (student)

Keywords: Memetic Algorithms, Images, Convolutional Deep Learning Models

Abstract

Deep Learning models have marked a true milestone in the field of **Artificial Intelligence**, as they are capable of processing large volumes of data and recognizing highly complex patterns. Among these, **convolutional models** have stood out as particularly effective in identifying objects and features in images—an essential capability for many modern applications. However, unlike humans, these models require a large amount of training data for each category they need to learn. This implies a longer training process, and in many cases, collecting the necessary data can be problematic depending on the type of information required.

In addition to the difficulty of obtaining data, new European regulations on artificial intelligence establish the need to audit not only the models but also the data used to train them, especially in AI applications that handle sensitive data. These audits, by their very nature, will become increasingly complex as the size of the training set grows. Therefore, it becomes essential to develop strategies that allow for **reducing the size of training datasets**, while minimizing any compromise in model quality.

In this work, we propose the use of **memetic algorithms** to implement a **training data reduction process**, also known as **instance selection**. The idea is to select a small set of representative images that, along with data augmentation techniques, are sufficient to optimally train convolutional models. In this way, it would be possible to significantly reduce the size of the training set while maintaining learning quality and, at the same time, facilitating both the auditing process and the system's computational efficiency.

Yo, **José Ruiz López**, alumno de la titulación INGENIERÍA INFORMÁTICA de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI **77964364E**, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: José Ruiz López

Granada a 24 de mayo de 2025.

D. **Daniel Molina Cabrera (tutor**, Profesor del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Algoritmos meméticos para reducir datos de entrenamiento en modelos de aprendizaje profundo convolucionales***, ha sido realizado bajo su supervisión por **José Ruiz López (alumno)**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 24 de mayo de 2025.

Los directores:

Daniel Molina Cabrera (tutor)

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que han hecho posible la realización de este Trabajo de Fin de Grado.

En primer lugar, me gustaría agradecer al profesor Daniel Molina Cabrera, mi tutor, por su valiosa guía, por su apoyo continuo durante todo el proceso y por brindarme acceso a los recursos necesarios para llevar a cabo esta investigación. Su experiencia y disponibilidad han sido fundamentales para que este proyecto pudiera desarrollarse de forma rigurosa y enriquecedora.

También quiero destacar lo desafiante que ha sido compaginar este trabajo académico con mis responsabilidades laborales. Ha requerido un esfuerzo constante y una gran capacidad de organización, pero también me ha permitido valorar aún más el proceso y el aprendizaje adquirido.

Agradezco de corazón a mis padres y amigos por su comprensión, paciencia y apoyo emocional durante los momentos más exigentes del camino. Y, sobre todo, a mi pareja, sin ella no habría sido posible superar el reto que supone realizar un trabajo como este. Su apoyo incondicional, su confianza en mí y la calma que me ha transmitido en los momentos más difíciles han sido clave para llegar hasta el final.

Asimismo, extiendo mi gratitud a todas las personas y compañeros que, directa o indirectamente, han contribuido con sus comentarios, sugerencias y tiempo. Gracias a ellos, este trabajo ha podido alcanzar una mayor profundidad y claridad.

Por último, quiero agradecer a todas aquellas comunidades de código abierto y herramientas libres que han permitido que este proyecto se desarrollara sin barreras tecnológicas, facilitando el aprendizaje y la innovación.

Índice general

Capítulo 1

Introducción

1.1. Contexto

En la actualidad, vivimos en una era marcada por una constante y acelerada generación de datos. Este fenómeno ha incrementado la necesidad de desarrollar métodos eficaces para el procesamiento y análisis de grandes volúmenes de información. En este contexto, los **modelos de aprendizaje profundo**, y en particular las **redes neuronales convolucionales** (CNN), han demostrado un notable rendimiento en tareas como la **clasificación de imágenes**, el **reconocimiento de patrones** y diversas aplicaciones de alta complejidad. No obstante, el entrenamiento de estos modelos suele requerir volúmenes elevados de datos, lo que plantea desafíos significativos tanto en términos de **tiempo** como de **costes** asociados a su obtención.

Conforme los sistemas de inteligencia artificial evolucionan hacia estructuras más sofisticadas y precisas, la disponibilidad de conjuntos de datos amplios y adecuados se vuelve un requisito cada vez más crucial. Sin embargo, la recopilación, almacenamiento y tratamiento de estos datos suponen obstáculos importantes, especialmente para aquellas instituciones u organizaciones que cuentan con recursos limitados. Esta situación pone de relieve la necesidad de investigar estrategias innovadoras que permitan **reducir y optimizar los conjuntos de datos** sin comprometer el rendimiento de los modelos entrenados.

1.2. Motivación

La necesidad de **reducir los conjuntos de datos de entrenamiento** responde al objetivo de mejorar la **eficiencia** en el desarrollo de modelos de aprendizaje profundo. Aunque las redes neuronales convolucionales han

demostrado un rendimiento notable en diversas tareas, su implementación conlleva **altos costes computacionales** y una gran demanda de datos, lo que supone un obstáculo importante en muchos escenarios, especialmente aquellos con recursos limitados. Una estrategia prometedora consiste en entrenar estos modelos utilizando únicamente una fracción de los datos disponibles, seleccionados de manera óptima. Esta aproximación permitiría disminuir significativamente el consumo de recursos sin comprometer la precisión del modelo, lo que supondría un avance importante para la **inteligencia artificial**, en particular en aplicaciones donde existen **restricciones de recursos**.

En este contexto, la selección inteligente de subconjuntos representativos constituye una estrategia eficaz para reducir tanto los tiempos de entrenamiento como el uso de recursos, sin afectar negativamente el rendimiento. Es precisamente en este punto donde las **metaheurísticas** adquieren un papel relevante. Estas técnicas de optimización están diseñadas para abordar problemas complejos en los que los métodos tradicionales resultan ineficaces, gracias a sus **estrategias de búsqueda** y **exploración del espacio de soluciones**. Al combinar diversas heurísticas, permiten encontrar soluciones aproximadas en tiempos razonables, lo que las convierte en una herramienta especialmente útil cuando la obtención de una solución exacta resulta inviable desde el punto de vista computacional.

Por tanto, las metaheurísticas se posicionan como una alternativa sólida para mejorar la eficiencia y accesibilidad del aprendizaje profundo, incluso en contextos con recursos limitados. Esto es especialmente relevante para avanzar hacia una inteligencia artificial más abierta, **democrática** y aplicable en una mayor variedad de escenarios.

Este Trabajo de Fin de Grado tiene como objetivo aplicar técnicas metaheurísticas para realizar una selección inteligente de ejemplos, con el fin de reducir el tamaño de los conjuntos de entrenamiento sin que ello afecte significativamente a los resultados obtenidos. La investigación aspira a contribuir al desarrollo de modelos más **eficientes**, accesibles y económicamente sostenibles, fomentando así un futuro en el que la inteligencia artificial sea más **inclusiva** y **sostenible**.

1.3. Objetivos

El objetivo principal de este TFG es investigar la aplicación de **metaheurísticas** para la **reducción de conjuntos de datos de entrenamiento** en modelos de **aprendizaje profundo convolucionales**. Este estudio permitirá evaluar el impacto de dichos algoritmos en la **eficiencia computacional** y en el **rendimiento de los modelos**. Para lograr estos objetivos,

se desarrollarán y compararán distintas técnicas de selección de instancias aplicadas sobre modelos convolucionales, incluyendo enfoques aleatorios, de búsqueda local, genéticos y meméticos.

Para cumplir con este objetivo general, se plantean los siguientes **objetivos específicos**:

- **Estudiar** la conveniencia del uso de metaheurísticas para la reducción de conjuntos de imágenes en modelos de aprendizaje profundo.
- **Desarrollar** y **mejorar** algoritmos metaheurísticos orientados a la selección de ejemplos representativos, con el objetivo de optimizar el entrenamiento de modelos convolucionales sin comprometer su rendimiento.
- **Evaluar** el impacto de la reducción de datos en el rendimiento de los modelos, comparando aspectos clave como la precisión, eficacia y el tiempo de entrenamiento, en modelos entrenados con conjuntos de datos completos frente a conjuntos reducidos.
- **Comparar** el rendimiento de metaheurísticas con porcentajes de selección fijos frente a aquellos que permiten una selección flexible del número de ejemplos, analizando sus ventajas y limitaciones.

A través de este estudio, se busca no solo mejorar el rendimiento y la eficiencia de los modelos convolucionales, sino también fomentar el desarrollo de soluciones más **sostenibles y accesibles** en el ámbito de la inteligencia artificial, mediante la incorporación de estrategias metaheurísticas como **propuestas innovadoras** para la reducción de datos de entrenamiento.

Capítulo 2

Metodología y Presupuesto

2.1. Metodología

Para organizar el desarrollo del proyecto se optó por una metodología inspirada en el marco ágil **Scrum** [1], ampliamente utilizado en entornos donde se requiere adaptación constante y entregas progresivas. Dado que el desarrollo del trabajo se ajustaba progresivamente según los resultados obtenidos en cada fase, este enfoque demostró ser adecuado para mantener una estructura flexible y favorecer un avance iterativo

2.1.1. Enfoque Ágil Basado en Scrum

El proyecto se dividió en ciclos de trabajo breves (**sprints**), generalmente de dos semanas. Cada sprint comenzaba con una planificación en la que se establecían objetivos concretos y finalizaba con una revisión para evaluar el progreso y hacer ajustes si era necesario. Esta dinámica permitió mantener un ritmo constante, al tiempo que se dejaba margen para adaptarse a posibles imprevistos o nuevas ideas surgidas durante el desarrollo.

2.1.2. Organización de Tareas

Para la organización y seguimiento de tareas se empleó una herramienta digital de apoyo (Notion [2]), que facilitó la planificación personal y visualización de actividades pendientes.

Este tipo de plataformas contribuyen a mejorar la organización y la colaboración en proyectos complejos, tal como se ha demostrado en entornos profesionales mediante el uso de tecnologías de la información orientadas a la gestión colaborativa [3].

Esta plataforma facilitó la visualización de las actividades pendientes y cumplidas, lo que ayudó a no perder de vista los plazos y prioridades de cada fase.

En paralelo, se empleó **Git** [4] para el control de versiones del código, lo que resultó fundamental para mantener un seguimiento detallado de los cambios realizados en cada etapa del proyecto. También se utilizó **GitHub** como repositorio central, facilitando así la colaboración con el tutor y permitiendo una trazabilidad clara de todas las modificaciones.

2.1.3. Ciclos de Trabajo

Los sprints se estructuraron en varias fases repetidas en cada iteración:

- **Planificación**: En esta etapa se definían los objetivos del sprint, basándose en lo ya realizado y en las tareas pendientes más prioritarias. El análisis del **backlog** permitía seleccionar actividades realistas que se pudieran completar en el plazo previsto.
- **Desarrollo e implementación**: Se ejecutaban las tareas previstas, como la programación de nuevos módulos, mejoras en algoritmos o la integración de componentes. El enfoque fue incremental, es decir, se añadían funcionalidades poco a poco, asegurando que cada avance se pudiera probar por separado.
- **Pruebas y ajustes**: Una vez desarrolladas las funcionalidades, se realizaban pruebas (unitarias, de integración o empíricas) para validar el funcionamiento del sistema y ajustar los parámetros si fuese necesario. A veces, los resultados de esta fase daban lugar a nuevas tareas que se añadían al backlog.
- **Revisión y análisis de resultados**: Al finalizar el sprint, se revisaban los objetivos cumplidos, se analizaban los resultados obtenidos y se valoraba la necesidad de replantear enfoques. Esta revisión también ayudaba a identificar puntos de mejora y reforzar los que ya funcionaban bien.
- **Documentación**: Durante todo el proceso se fue registrando la evolución del proyecto, desde los cambios en el código hasta los resultados de las pruebas. Este esfuerzo permitió tener una memoria bien estructurada, facilitar la reproducción de experimentos y justificar cada decisión tomada.

2.1.4. Seguimiento del Progreso

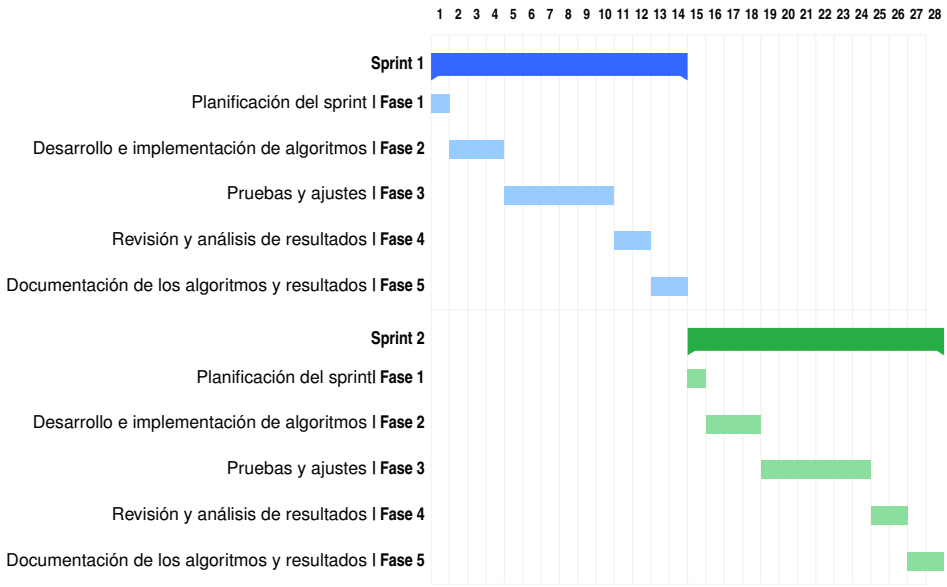

Figura 2.1: Diagrama de Gantt que muestra la planificación de dos sprints del proyecto, con sus respectivas fases: planificación, desarrollo, pruebas, revisión y documentación. Este esquema representa la estructura iterativa adoptada a lo largo del desarrollo

Para tener una visión clara del avance del proyecto, se elaboró un **diagrama de Gantt** [2.1] que recogía dos sprints como ejemplo del proceso general. En él se reflejan las distintas fases mencionadas, así como la duración aproximada de cada una. Aunque se realizaron más sprints a lo largo del trabajo, este diagrama sirve como muestra del esquema de planificación utilizado.

2.1.5. Ajuste de los Tiempos

Durante el desarrollo surgieron algunos retrasos que obligaron a reajustar los tiempos previstos para los sprints. Gracias a la flexibilidad que permite **Scrum** y al uso de herramientas como **Git**, permitió reorganizar prioridades y redistribuir tareas sin comprometer los objetivos del proyecto sin comprometer la calidad del trabajo.

2.2. Presupuesto

El presupuesto estimado incluye tanto el tiempo dedicado al proyecto (mano de obra) como los recursos computacionales y otros gastos relacionados. Aunque muchos recursos utilizados son gratuitos o de bajo coste, se ha realizado una estimación considerando el valor del tiempo invertido y el uso de recursos computacionales.

2.2.1. Mano de obra

El proyecto ha requerido un esfuerzo considerable distribuido entre tareas de análisis, desarrollo e implementación técnica, así como documentación y validación experimental. En total, se estima que se han dedicado aproximadamente **400 horas** al trabajo, repartidas de forma flexible a lo largo del calendario del TFG.

La estimación del coste de la mano de obra se ha calculado a partir de una tarifa de **20 euros** por hora, en línea con valores orientativos para trabajos técnicos similares [5].

A continuación se detallan las tareas principales:

- **Planificación inicial y diseño del proyecto**: Establecimiento de objetivos, enfoque metodológico y cronograma.
- **Repaso bibliográfico**: Revisión de trabajos previos relacionados con algoritmos meméticos y reducción de datos en aprendizaje profundo.
- **Comprensión del problema y formulación**: Definición precisa del problema y formalización de los objetivos computacionales.
- **Selección de datasets y preprocesado**: Búsqueda, limpieza y organización de los conjuntos de datos utilizados.
- **Implementación inicial de algoritmos**: Desarrollo de versiones base de los algoritmos genético, memético y de búsqueda local.
- **Desarrollo de versiones mejoradas**: Ajustes iterativos, experimentación con nuevas variantes y mecanismos de exploración.
- **Evaluación experimental y análisis de resultados**: Ejecución masiva de pruebas, análisis comparativo y visualización de métricas.
- **Generación de gráficos**: Creación de boxplots, barplots y figuras de evolución del fitness.

- **Escritura de la memoria del TFG**: Redacción estructurada del documento final, incluyendo capítulos técnicos, resultados y conclusiones.
- **Revisión y corrección**: Mejora del contenido tras la retroalimentación del tutor, ajuste de redacción, formato y bibliografía.

Tarea realizada	**Horas dedicadas**	**Coste por hora (€)**	**Coste total (€)**
Planificación inicial y diseño del proyecto	20	20	400
Repaso bibliográfico	30	20	600
Comprensión del problema y formulación	25	20	500
Selección de datasets y preprocesado inicial	20	20	400
Implementación inicial de algoritmos	40	20	800
Desarrollo de propuestas y versiones mejoradas	50	20	1.000
Evaluación experimental y análisis de resultados	60	20	1.200
Generación de gráficos	30	20	600
Escritura de la memoria del TFG	80	20	1.600
Revisión y corrección final del documento	45	20	900
Total	**400 horas**		**8.000 €**

Tabla 2.1: Estimación detallada del coste de la mano de obra distribuido por tareas del proyecto.

A modo de resumen, en la Tabla 2.1 se presenta una distribución detallada de las horas dedicadas a cada una de estas tareas, junto con el coste estimado asociado. Esta descomposición permite visualizar de forma clara cómo se ha estructurado el esfuerzo invertido, diferenciando tareas de investigación, desarrollo técnico y documentación. El resultado total asciende a 8.000 €, lo que proporciona una visión aproximada del valor económico del trabajo llevado a cabo.

2.2.2. Recursos computacionales

El proyecto ha requerido el uso de recursos computacionales para entrenar los modelos de aprendizaje profundo, especialmente para evaluar la eficacia de los algoritmos en la reducción de datos.

El tutor **Daniel Molina Cabrera** me puso en disposición el acceso a un servidor de investigación con disponibilidad de GPU, de manera que no ha supuesto ningún coste.

Aunque el acceso al servidor de investigación con GPU no implicó coste real, se ha realizado una estimación hipotética utilizando precios de **Google Cloud** [6] usando un **Compute Engine** [7] en Bélgica para valorar el uso de recursos.

Los componentes equivalentes del servidor en Compute Engine serían:

- Un **Intel Xeon E-2226G** equivaldría a un **c2-standard-8**, cuyo coste es de 0.43 EUR/h.

- Un **NVIDIA TITAN Xp** equivaldría a una **NVIDIA K80 1 GPU**, GDDR5 de 12 GB cuyo coste es de 0.42 EUR/h.

De modo que los gastos estimados serían:

Recurso	Horas utilizadas	Coste por hora (€)	Coste total (€)
CPU (c2-standard-8)	600	0.43	258
GPU (NVIDIA K80 1 GPU)	600	0.42	252
Total	**600**	**0.85**	**510**

Tabla 2.2: Simulación del coste de recursos computacionales equivalentes en Google Cloud.

2.2.3. Software y licencias

Para este proyecto, todas las herramientas de desarrollo utilizadas han sido de **código abierto**, por lo que no se han incurrido en costes de licencias.

2.2.4. Gastos indirectos

Aunque muchos de los recursos utilizados durante el desarrollo del proyecto han sido gratuitos o provistos por el entorno universitario, se ha considerado el coste indirecto asociado al uso continuado del equipo personal.

En concreto, se estima el coste proporcional derivado del uso de un ordenador portátil **Lenovo Ideapad 530S-14KB**, empleado intensivamente durante todo el curso académico para la programación, análisis y documentación del TFG. El cálculo se ha realizado prorrateando su valor de adquisición estimado (aproximadamente 900 €) a lo largo de una vida útil de 5 años, lo que da lugar a un coste anual aproximado de 180 €.

Concepto	Coste estimado (€)
Uso del portátil (Lenovo Ideapad 530S)	180
Total	**180 €**

Tabla 2.3: Estimación de gastos indirectos asociados al desarrollo del proyecto.

2.2.5. Presupuesto total

Sumando los costes de mano de obra, los recursos computacionales y otros gastos, el presupuesto total estimado es el siguiente:

Concepto	**Coste (€)**
Mano de obra	8.000
Recursos computacionales	510
Gastos indirectos	180
Total	**8.690 €**

Tabla 2.4: Resumen del presupuesto total estimado, incluyendo mano de obra, recursos y gastos indirectos.

En conjunto, la planificación metodológica y la estimación de recursos permitieron garantizar el avance ordenado del proyecto y anticipar su viabilidad técnica y económica.

Capítulo 3

Fundamentos de Aprendizaje Profundo

El presente capítulo tiene como objetivo proporcionar una base teórica sólida sobre los conceptos clave del **aprendizaje profundo**, que constituyen el núcleo metodológico del presente Trabajo de Fin de Grado. Dado que el proyecto se centra en optimizar el entrenamiento de modelos de redes neuronales convolucionales mediante técnicas de reducción de datos, resulta fundamental comprender el funcionamiento interno de este tipo de modelos y sus componentes principales.

Esta revisión técnica permitirá comprender mejor los retos que supone entrenar dichos modelos y la motivación detrás de aplicar algoritmos meméticos para optimizar el proceso.

3.1. Definición de Aprendizaje Profundo

El **aprendizaje profundo** (Deep Learning) [8] es una subcategoría del aprendizaje automático que se basa en el uso de **redes neuronales artificiales** con muchas capas (de ahí el término "profundo"). Estas redes están diseñadas para imitar el funcionamiento del cerebro humano, lo que les permite aprender representaciones complejas de los datos de manera jerárquica.

La principal diferencia entre el **aprendizaje automático tradicional** y el aprendizaje profundo es la manera en que se manejan las características de los datos:

- En los enfoques tradicionales, el ingeniero o científico de datos debe extraer manualmente las características más importantes para entrenar al modelo (por ejemplo, bordes, formas, texturas en imágenes). En el

aprendizaje profundo, las redes neuronales son capaces de **aprender automáticamente las representaciones de los datos** a partir de los datos crudos (por ejemplo, imágenes, texto, sonido).

- Este proceso es denominado **aprendizaje de características** (feature learning), lo que reduce la necesidad de intervención humana.

El aprendizaje profundo ha mostrado un rendimiento sobresaliente en diversas tareas, como el reconocimiento de imágenes, el procesamiento del lenguaje natural, la conducción autónoma y el diagnóstico médico, gracias a su capacidad para **capturar patrones complejos** en grandes volúmenes de datos.

3.2. Redes Neuronales Artificiales

Las **redes neuronales artificiales** (ANN) [9] son el corazón del aprendizaje profundo. Estas redes están compuestas por neuronas artificiales, que son unidades matemáticas inspiradas en las neuronas biológicas. Cada neurona toma varias entradas, las procesa mediante una **función de activación**, y produce una salida. Cuando se combinan muchas de estas neuronas en capas, forman una red neuronal.

3.2.1. Componentes de una Red Neuronal

1. **Neuronas o Unidades**:

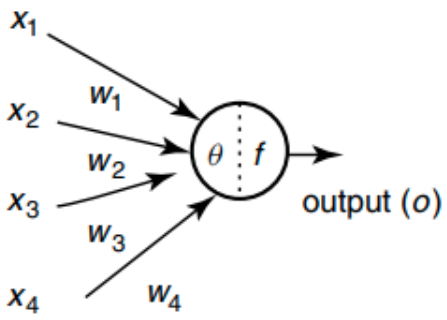

Figura 3.1

Cada **neurona** realiza una operación simple: recibe varias entradas, las pondera por medio de **pesos** w_i, suma estos valores junto con un

sesgo b, y aplica una función de activación. La salida de la neurona se expresa como:

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b_z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

Luego, el valor z pasa por una función de activación, que introduce la no linealidad en el sistema, permitiendo que las redes neuronales modelen relaciones complejas.

2. **Capas de la Red**:

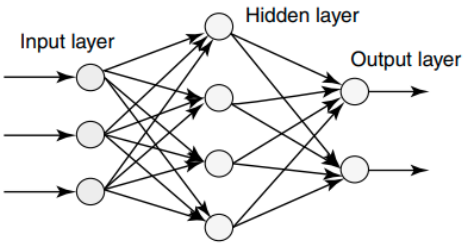

Figura 3.2

   - **Capa de entrada**: Es la primera capa de la red neuronal, que recibe los datos crudos (por ejemplo, píxeles de una imagen).
   - **Capas ocultas**: Estas capas intermedias entre la entrada y la salida aprenden representaciones abstractas de los datos. En una red profunda, hay múltiples capas ocultas, lo que permite la **transformación jerárquica** de los datos.
   - **Capa de salida**: Produce la predicción final, que puede ser una clase (en problemas de clasificación) o un valor numérico (en problemas de regresión).

3. **Pesos y Bias**: Los **pesos** son parámetros ajustables que determinan la importancia de cada entrada en la neurona. El **bias** es otro parámetro que se suma al valor ponderado para desplazar la activación de la neurona y permitir que el modelo ajuste mejor los datos.

4. **Funciones de Activación**: Las funciones de activación son fundamentales para que las redes neuronales puedan aprender relaciones no lineales. Entre las más comunes se encuentran:

   - **ReLU(Rectified Linear Unit)**: $ReLU(x) = \text{máx}(0, x)$, que activa solo valores positivos.
   - **Sigmoide**: Que transforma los valores en un rango entre 0 y 1.
   - **Tanh (Tangente hiperbólica)**: Transforma los valores en un rango entre -1 y 1.

El uso de **backpropagation** o retropropagación permite ajustar los pesos y biases durante el entrenamiento mediante un algoritmo de optimización, como el descenso de gradiente. De esta manera, la red aprende minimizando la diferencia entre sus predicciones y las respuestas correctas.

3.3. Redes Neuronales Convolucionales

Las **Redes Neuronales Convolucionales** (Convolutional Neural Networks, CNN) son una clase de redes neuronales profundas especialmente efectivas para el procesamiento de datos que tienen una estructura de tipo rejilla, como las imágenes. Fueron inspiradas por el sistema visual de los mamíferos, donde diferentes capas de neuronas responden a estímulos visuales de manera jerárquica.

Las CNN son ampliamente utilizadas en tareas de **visión por computador**, como el reconocimiento de imágenes, la segmentación de objetos y la clasificación de imágenes. Lo que diferencia a las CNN de las redes neuronales tradicionales es su capacidad para detectar **patrones espaciales** como bordes, texturas, y formas, sin necesidad de un procesamiento manual de las características.

3.3.1. Componentes principales de una CNN

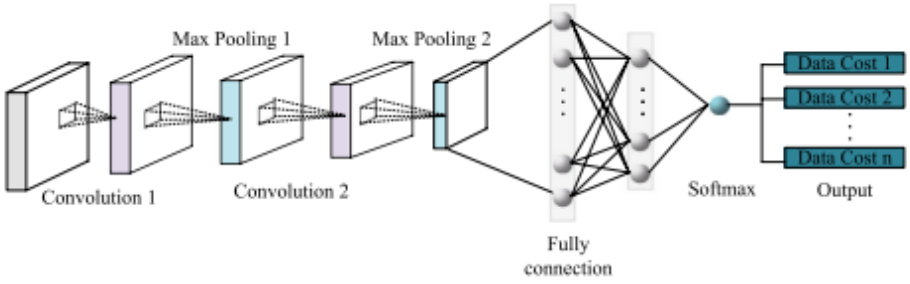

Figura 3.3: En esta imagen extraída de [10] puede observarse de la estructura de una red neuronale convolucional. Junto a sus capas convolucionales.

1. **Capas Convolucionales**: Estas capas aplican **filtros o kernels** sobre las imágenes de entrada para detectar características locales, como bordes, esquinas o texturas. Un filtro convolucional es una pequeña matriz que se mueve a lo largo de la imagen, calculando productos escalares en cada posición para producir un mapa de características.

Las convoluciones son útiles porque explotan la **localidad de las características**, es decir, las relaciones espaciales entre píxeles cercanos. Además, la cantidad de parámetros se reduce drásticamente en comparación con las capas densas, ya que el filtro se comparte a lo largo de la imagen.

2. **Pooling (Submuestreo o Agrupamiento)**: Las capas de pooling reducen la dimensionalidad de las características extraídas por las capas convolucionales, lo que hace que las representaciones sean más manejables y robustas frente a pequeños cambios o desplazamientos en la imagen.

   El max-pooling es la técnica de pooling más común, donde se toma el valor máximo dentro de una ventana de píxeles, reduciendo el tamaño de la imagen, pero reteniendo las características más importantes.

3. **Capas Densas**: Después de varias capas convolucionales y de pooling, se agregan una o más **capas densas** (fully connected) para realizar la clasificación o predicción final. Estas capas toman todas las características aprendidas en las capas convolucionales y las combinan para generar una decisión final.

4. **Batch Normalization**: Esta técnica se utiliza para **normalizar** las salidas de las capas intermedias de una red neuronal. Batch Normalization ayuda a **acelerar el entrenamiento** y a hacer que la red sea más estable, al reducir el **desplazamiento covariante** (cambios en las distribuciones de las entradas de las capas intermedias a lo largo del entrenamiento). Esto se logra al normalizar las entradas de cada capa convolucional o densa antes de aplicar la activación, ajustando su media y varianza.

5. **Dropout**: El Dropout es una técnica de **regularización** que se utiliza para prevenir el **sobreajuste** (overfitting) durante el entrenamiento de una red neuronal. Durante cada iteración del entrenamiento, Dropout **desactiva aleatoriamente** un porcentaje de las neuronas, lo que obliga a la red a no depender excesivamente de ciertas neuronas y a ser más robusta. Esta técnica mejora la generalización de la red, lo que la hace funcionar mejor en datos no vistos.

3.3.2. Funcionamiento General de una CNN

Al pasar una imagen a través de varias capas convolucionales, la red aprende a identificar características simples como líneas y bordes. Conforme avanza a capas más profundas, las características se vuelven más abstractas, capturando patrones más complejos como formas, texturas y, finalmente, estructuras completas como objetos.

Por ejemplo, en una red entrenada para reconocer caras, las primeras capas pueden detectar bordes o contornos, las capas intermedias pueden aprender a reconocer ojos, nariz o boca, y las últimas capas pueden identificar una cara completa.

3.3.3. Aplicaciones de las CNN

- **Clasificación de imágenes**: Etiquetar imágenes en distintas categorías, como identificar animales o vehículos.
- **Detección de objetos**: Identificar y localizar objetos en imágenes.
- **Reconocimiento facial**: Utilizado en sistemas de seguridad, como el desbloqueo de teléfonos móviles.

Las CNN son fundamentales en muchas aplicaciones modernas debido a su capacidad para procesar y entender datos visuales de manera eficiente y automática.

3.4. Modelos

En el ámbito del aprendizaje profundo, existen diversas arquitecturas de redes neuronales convolucionales que han demostrado un rendimiento excepcional en diversas tareas de visión por computadora. Estas arquitecturas están diseñadas para abordar problemas complejos y variados, desde la clasificación de imágenes hasta la detección de objetos y el segmentado de imágenes.

A continuación, se explorarán algunas de estas arquitecturas que representan avances significativos en la eficiencia y efectividad del aprendizaje profundo.

3.4.1. ResNet50

ResNet50 [11] es una arquitectura de red neuronal convolucional introducida por `Kaiming He et` [12]. La principal innovación de ResNet es la introducción de **bloques de residualidad**, que permiten la construcción de redes extremadamente profundas sin el problema de la degradación del rendimiento.

La idea básica de los bloques residuales es permitir que la red aprenda funciones de identidad, facilitando así la propagación de la información y el gradiente a través de la red. En términos prácticos, esto se traduce

en un rendimiento mejorado en tareas de clasificación de imágenes, donde ResNet50 ha logrado resultados sobresalientes en competiciones como ImageNet [13]. Esta arquitectura se compone de 50 capas, de las cuales 49 son convolucionales y una es totalmente conectada, incluyendo capas de normalización y activación (ReLU), así como conexiones residuales que facilitan el entrenamiento de redes profundas.

3.4.2. MobileNetV2

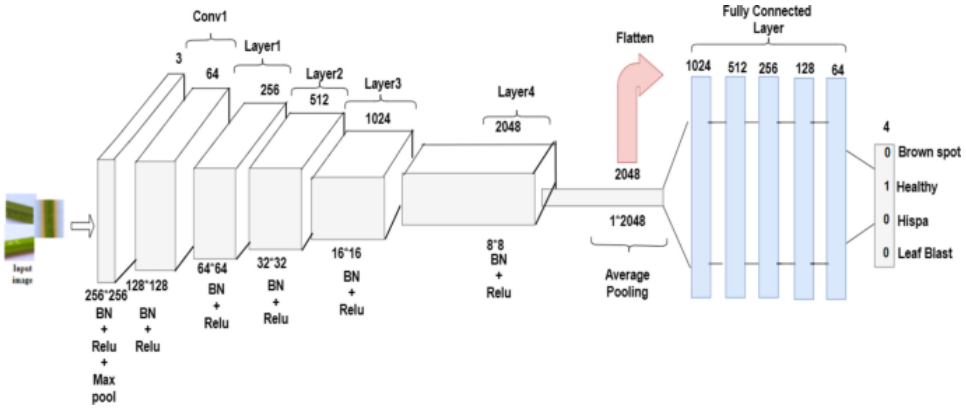

Figura 3.4: Diagrama de la Arquitectuda de ResNet50

MobileNet [14] es una arquitectura de red neuronal diseñada específicamente para aplicaciones móviles y de visión por computadora en dispositivos con recursos limitados. Introducida por Andrew G. Howard et al. en 2017, MobileNet se basa en el principio de **convoluciones separables en profundidad** (depthwise separable convolutions), que dividen el proceso de convolución en dos pasos: primero, se aplica una convolución a cada canal de la entrada (depthwise), y luego, se combinan los resultados con una convolución 1x1 (pointwise).

Esta técnica reduce significativamente el número de parámetros y el costo computacional, lo que permite ejecutar modelos de visión por computadora en dispositivos móviles sin sacrificar drásticamente la precisión.

MobileNetV2

MobileNetV2 [15], introducido por Sandler et al. en 2018, mejora la arquitectura de MobileNet original al incorporar varias innovaciones. La principal contribución de MobileNetV2 es la introducción de los bloques de **residualidad invertida** (inverted residuals), que permiten que la red

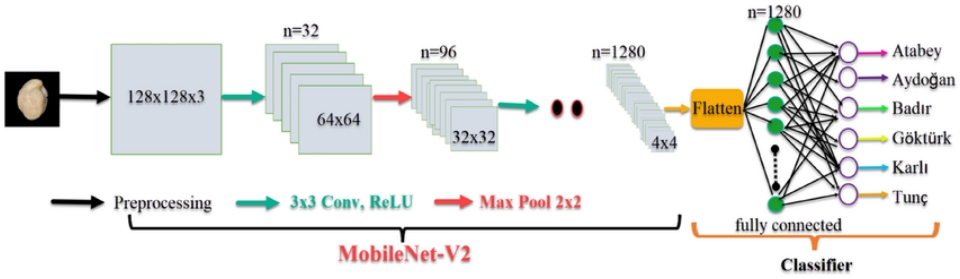

Figura 3.5: Diagrama de la Arquitectuda de MobileNetV2

mantenga una mayor capacidad de representación y flujo de información a través de las capas.

Además, MobileNetV2 utiliza una función de activación llamada **linear bottleneck**, que ayuda a preservar la información durante la propagación a través de las capas, lo que mejora aún más el rendimiento del modelo en tareas de clasificación y detección. Esta arquitectura se optimiza para ser altamente eficiente, permitiendo que sea utilizada en aplicaciones de tiempo real en dispositivos con limitaciones de hardware.

MobileNetV2 ha demostrado ser una opción popular para aplicaciones en dispositivos móviles y sistemas embebidos, ofreciendo un buen equilibrio entre precisión y eficiencia computacional.

Capítulo 4

Repaso Bibliográfico

En este capítulo se revisa la literatura relevante en torno a los modelos de **aprendizaje profundo**, la necesidad de reducir los conjuntos de **datos de entrenamiento** y el rol de los **algoritmos meméticos** en esta tarea. A lo largo de este capítulo, se analizan estudios previos que aplican técnicas de **optimización** para reducir los datos necesarios en el entrenamiento de **redes neuronales convolucionales** (CNN) y se exploran las soluciones existentes en la selección de instancias mediante algoritmos **evolutivos** y **meméticos**.

4.1. Reducción de Conjuntos de Datos en Aprendizaje Profundo

El **aprendizaje profundo**, especialmente en el campo de la **visión por computadora**, se ha basado en grandes volúmenes de **datos** para alcanzar su éxito. En particular, las redes neuronales convolucionales (CNN) han demostrado un rendimiento notable en tareas de **clasificación**, **detección** y **segmentación** de imágenes [16]. Sin embargo, el volumen de datos necesario para entrenar estos modelos supone retos en cuanto a la disponibilidad de **almacenamiento**, el **tiempo de entrenamiento** y el **costo computacional**, especialmente en sistemas con **recursos limitados** [17].

La necesidad de reducir estos volúmenes de datos sin comprometer la **precisión** del modelo ha impulsado investigaciones en torno a técnicas de **selección de instancias**, donde el objetivo es identificar y mantener solo las instancias de datos que aportan valor al modelo. A este respecto, las técnicas de selección de instancias combinadas con estrategias de **aumento de datos** [18] permiten reducir el tamaño de los conjuntos de datos manteniendo una **diversidad** adecuada, lo cual es crucial para evitar el **so-**

breajuste y la pérdida de **generalización** en las redes neuronales. Además, otros enfoques, como el **submuestreo de datos** y la generación de **conjuntos sintéticos**, se han propuesto como soluciones complementarias en el contexto de aprendizaje profundo, siendo útiles en escenarios con conjuntos de datos desequilibrados o insuficientes.

4.2. Selección de Instancias mediante Algoritmos Evolutivos y Meméticos

La **selección de instancias** es una técnica de reducción de datos que se centra en eliminar aquellas instancias **redundantes** o **irrelevantes**, mejorando la **eficiencia** y manteniendo la **precisión** en el modelo entrenado. Los **algoritmos evolutivos**, como los **algoritmos genéticos**, han demostrado eficacia en la selección de instancias al simular procesos de **selección natural**, con la ventaja de que pueden explorar un espacio de **soluciones** de manera eficiente a través de **operadores genéticos** como la **selección**, el **cruce** y la **mutación** [19].

Dentro de este campo, los **algoritmos meméticos** ofrecen un avance significativo al combinar los principios de **optimización evolutiva** con técnicas de **búsqueda local**, lo que permite una adaptación más precisa de las instancias seleccionadas [20]. Estos algoritmos representan un enfoque **híbrido**, ya que aprovechan la capacidad **exploratoria** de los algoritmos evolutivos con la capacidad **explotatoria** de las búsquedas locales, lo cual es crucial para alcanzar una **convergencia óptima**. Al introducir esta **dualidad**, los algoritmos meméticos pueden enfocarse en subconjuntos de datos más representativos y potencialmente relevantes para el modelo, maximizando la reducción sin comprometer la calidad del aprendizaje.

4.3. Algoritmos Meméticos en Modelos Convolucionales

La capacidad de los **algoritmos meméticos** para optimizar la selección de datos se ha aplicado a **modelos convolucionales**, en particular en contextos de redes profundas que requieren grandes cantidades de datos para alcanzar un rendimiento óptimo [21]. La selección de instancias mediante algoritmos meméticos permite mantener la precisión del modelo, pero con un volumen de datos significativamente reducido, lo cual es especialmente útil en aplicaciones donde los recursos computacionales y el tiempo son limitados [22].

Algunos estudios han demostrado que los algoritmos meméticos no solo

reducen los tiempos de **entrenamiento** y el tamaño de los conjuntos de **datos**, sino que también ayudan a minimizar el riesgo de **sobreajuste**. Al reducir los datos redundantes, el modelo **convolucional** puede centrarse en **patrones** más específicos, potenciando su capacidad de **generalización** y **robustez**. Estas ventajas son especialmente valiosas en escenarios como el **reconocimiento facial** o la **medicina**, donde la precisión y la eficiencia en el procesamiento de imágenes son esenciales.

4.4. Aplicaciones y Desafíos de los Algoritmos Meméticos

A pesar de sus beneficios, la aplicación de **algoritmos meméticos** en modelos de aprendizaje profundo plantea desafíos. Uno de los principales es la necesidad de ajustar **parámetros complejos**, como el tamaño de la **población**, las tasas de **mutación** y los métodos de **selección** y **cruce**. Estos parámetros afectan de manera directa la **velocidad de convergencia** y la capacidad del algoritmo para encontrar soluciones **óptimas**. Además, los algoritmos meméticos suelen ser computacionalmente **exigentes**, lo que plantea una paradoja cuando se aplican en entornos con **recursos limitados** [23].

Para abordar estos desafíos, investigaciones recientes han explorado el desarrollo de **variantes de algoritmos meméticos adaptativos**, que ajustan automáticamente sus parámetros en función del desempeño durante el proceso de **entrenamiento**. Esta **adaptabilidad** representa una vía prometedora, ya que permite que los algoritmos se adapten dinámicamente a los cambios en el entorno de datos y en las necesidades del modelo, facilitando así su implementación en aplicaciones prácticas.

4.5. Perspectivas Futuras en la Optimización de CNN con Algoritmos Meméticos

La combinación de **CNN** y **algoritmos meméticos** sigue siendo un área emergente y prometedora en la investigación de **redes neuronales profundas**. A medida que la tecnología y la **capacidad de procesamiento** evolucionan, es probable que los algoritmos meméticos se integren de manera más fluida en **arquitecturas de aprendizaje profundo**, no solo para la selección de instancias, sino también para la **optimización de hiperparámetros** y **estructuras de red**. Asimismo, se anticipa que la investigación futura también explore la combinación de estos métodos con otros enfoques avanzados de **reducción de datos**, como las técnicas de

distilación de modelos y la **poda de redes neuronales**.

En conclusión, este capítulo ha establecido las bases teóricas que sustentan la selección de **algoritmos meméticos** para reducir conjuntos de datos en **redes profundas**, resaltando su relevancia en la optimización de **modelos CNN** en entornos de recursos limitados.

Capítulo 5

Descripción de los Algoritmos

En este capítulo se describen los diferentes algoritmos utilizados en el desarrollo de este trabajo. Todos ellos tienen como objetivo principal reducir el tamaño del conjunto de datos de entrenamiento utilizado en los modelos de aprendizaje profundo, con el fin de optimizar el rendimiento y reducir el costo computacional. El enfoque adoptado en este trabajo es la aplicación de algoritmos meméticos, los cuales combinan principios de algoritmos genéticos con estrategias de búsqueda local.

A continuación, se detallan los algoritmos principales implementados en este proyecto: el **algoritmo aleatorio**, el **algoritmo de búsqueda local**, el **algoritmo genético**y el **algoritmo memético**.

5.1. Algoritmo Aleatorio

El **algoritmo aleatorio** sirve como referencia básica para medir la efectividad de los algoritmos más avanzados. Este enfoque selecciona subconjuntos de datos de manera completamente aleatoria, sin aplicar ningún tipo de estrategia de optimización.

5.1.1. Descripción

El algoritmo comienza tomando el conjunto de datos completo y seleccionando una fracción de los ejemplos de entrenamiento de forma aleatoria. Esta selección se realiza sin ningún criterio basado en la relevancia de los datos, lo que implica que el conjunto de entrenamiento resultante puede no ser representativo o puede contener redundancias innecesarias.

5.1.2. Pseudocódigo

Algorithm 1 Algoritmo Aleatorio

```
Inicializar evaluaciones ← 0
while evaluaciones <máximo do
    Generar una selección aleatoria
    Evaluar la selección
    if mejora el mejor fitness global then
        Guardar como mejor solución
        Reiniciar contador sin mejora
    else
        Incrementar contador sin mejora
    end if
    Incrementar número de evaluaciones
    if se supera el umbral de estancamiento then
        Terminar la búsqueda
    end if
end while
return mejor solución e historial
```

Para facilitar su lectura o replicación, se ha detallado el pseudocódigo del **Algorithm 1** Algoritmo Aleatorio.

5.1.3. Aplicación en la reducción de datos

A pesar de su simplicidad, el **algoritmo aleatorio** puede ser útil como método de comparación. En muchos casos, los algoritmos más complejos deben demostrar que pueden superar este enfoque básico en términos de precisión y eficiencia. Al seleccionar datos de manera aleatoria, este método a menudo produce conjuntos de entrenamiento subóptimos, lo que resulta en modelos menos precisos o con mayor varianza.

5.1.4. Resultados esperados

Debido a la naturaleza aleatoria del algoritmo, los resultados son altamente variables. Es probable que en muchas ejecuciones el rendimiento del modelo entrenado sea inferior al obtenido con métodos más estructurados. Este algoritmo proporciona una línea base importante para evaluar la efectividad de los algoritmos más avanzados.

5.2. Algoritmo Búsqueda Local

El **algoritmo de búsqueda local** es una técnica más sofisticada que explora el espacio de soluciones de manera más estructurada, buscando mejorar progresivamente una solución inicial.

5.2.1. Descripción

La búsqueda local se basa en la idea de comenzar con una solución inicial (un subconjunto de datos) y realizar pequeños cambios o 'movimientos' en esa solución para explorar otras soluciones cercanas. En este contexto, cada solución es un subconjunto de datos. El algoritmo evalúa diferentes subconjuntos de datos probando si estos mejoran el rendimiento del modelo de aprendizaje profundo al entrenarlo con ellos.

5.2.2. Pseudocódigo

Algorithm 2 Algoritmo de Búsqueda Local

```
Generar una solución inicial aleatoria
Evaluar su fitness como mejor_fitness
while evaluaciones < máximo do
    Generar un vecino de la solución actual
    Evaluar el fitness del vecino
    if el vecino mejora o iguala la solución actual then
        Reemplazar la solución actual
        if mejora el mejor_fitness then
            Actualizar mejor solución
            Reiniciar contador sin mejora
        else
            Incrementar contador sin mejora
        end if
    else
        Incrementar contador sin mejora
    end if
    Incrementar evaluaciones
    if hay estancamiento then
        Terminar búsqueda
    end if
end while
return mejor solución
```

Para facilitar su lectura o replicación, se ha detallado el pseudocódigo

del **Algorithm 2** Algoritmo Búsqueda Local.

5.2.3. Aplicación en la reducción de datos

En el contexto de la reducción de datos, el objetivo de la búsqueda local es identificar un subconjunto más pequeño de ejemplos que sea suficiente para entrenar el modelo con un rendimiento similar al obtenido con el conjunto de datos completo. La búsqueda local explora el espacio de posibles subconjuntos, eliminando ejemplos redundantes o irrelevantes, y conservando solo aquellos que son cruciales para el rendimiento del modelo.

5.2.4. Ventajas y limitaciones

Ventajas: Este enfoque permite una exploración más exhaustiva del espacio de soluciones que un algoritmo aleatorio. Al hacer pequeños ajustes en cada iteración, el algoritmo puede encontrar mejores soluciones de manera eficiente.

Limitaciones: Sin embargo, la búsqueda local puede quedarse atrapada en **óptimos locales**, es decir, soluciones que parecen buenas en comparación con las cercanas, pero que no son globalmente óptimas.

5.3. Algoritmo Genético

5.3.1. Descripción

El algoritmo genético es una técnica de optimización inspirada en los principios de la evolución natural. Parte de una población inicial de soluciones candidatas (en este caso, subconjuntos de imágenes) que evoluciona generación tras generación mediante operadores de selección, cruce y mutación. Cada una de estas soluciones se evalúa con respecto a su aptitud (fitness), que se calcula según la métrica de rendimiento del modelo entrenado con dicho subconjunto (accuracy).

En esta versión inicial, el algoritmo utiliza una implementación sencilla pero funcional, diseñada para validar los beneficios de la evolución artificial sobre enfoques aleatorios o deterministas. A pesar de su simplicidad, esta versión ya logra una exploración significativa del espacio de soluciones, sirviendo como base para versiones posteriores más sofisticadas.

5.3.2. Componentes del Algoritmo

A continuación, se describen los principales componentes que conforman esta primera versión del algoritmo genético:

Población inicial

Se genera aleatoriamente una colección de subconjuntos de datos, donde cada individuo representa una selección binaria de imágenes. Este conjunto inicial sirve como punto de partida para la evolución del sistema.

Evaluación

Cada individuo se evalúa entrenando un modelo convolucional sobre el subconjunto de imágenes representado. El valor de *fitness* se calcula utilizando la métrica *accuracy*.

Selección por torneo

Se escogen dos padres de la población mediante el método de torneo. Para ello, se elige aleatoriamente un subconjunto de individuos y se selecciona el que tenga mayor fitness. Este proceso simula una competencia entre soluciones y prioriza las más prometedoras.

Cruce

Se aplica un operador de cruce básico que intercambia parte de las imágenes entre ambos padres. Este operador garantiza que el número total de imágenes seleccionadas en los hijos se mantenga constante, preservando la estructura del problema.

Mutación

Cada hijo generado puede sufrir mutaciones aleatorias. Con una probabilidad fija, algunas imágenes cambian su estado (de seleccionadas a no seleccionadas y viceversa). Este operador introduce diversidad genética en la población, evitando el estancamiento evolutivo.

Elitismo

Para asegurar que no se pierdan buenas soluciones, el mejor individuo de cada generación se conserva sin modificación. Esta estrategia protege el progreso evolutivo alcanzado hasta el momento.

Criterio de parada

El algoritmo se detiene cuando se alcanza un número máximo de evaluaciones o si no se ha logrado mejorar el mejor fitness durante un número consecutivo de iteraciones. Esto evita ciclos innecesarios y mejora la eficiencia computacional.

5.3.3. Pseudocódigo

Algorithm 3 Algoritmo Genético

```
1: Inicializar población aleatoria
2: Evaluar fitness de cada individuo
3: Guardar el mejor individuo
4: while no se alcance el número máximo de evaluaciones do
5:     Aplicar elitismo
6:     while población incompleta do
7:         Seleccionar dos padres por torneo
8:         Cruzar padres para obtener hijos
9:         Mutar hijos
10:        Evaluar hijos y añadirlos a la nueva población
11:    end while
12:    Reemplazar población
13:    Actualizar mejor individuo si mejora
14: end while
15: return mejor solución
```

Para facilitar su lectura o replicación, se ha detallado el pseudocódigo del **Algorithm 3** Algoritmo Genético.

5.3.4. Aplicación en la Reducción de Datos

El algoritmo genético se adapta de forma natural a la tarea de reducción de datos, ya que cada individuo puede representar explícitamente qué instancias se incluyen o excluyen del conjunto de entrenamiento. Esta representación binaria permite una manipulación sencilla de los subconjuntos y favorece la exploración combinatoria de múltiples configuraciones posibles.

Gracias a su enfoque evolutivo, este algoritmo puede descubrir subconjuntos de imágenes que, aunque significativamente más pequeños que el conjunto completo, mantienen un alto rendimiento del modelo. Además, el uso de operadores estocásticos contribuye a escapar de soluciones triviales o subóptimas.

5.3.5. Ventajas y Limitaciones

Ventajas:

- Flexible y fácilmente ajustable mediante cambios en los operadores.
- Capaz de encontrar buenas soluciones en espacios grandes y discretos.
- Evoluciona de forma progresiva, permitiendo un seguimiento del proceso.

Limitaciones:

- El operador de cruce simple puede resultar limitado, ya que no considera la calidad relativa de los padres.
- Sensible a la convergencia prematura si la diversidad de la población se pierde rápidamente.
- Aunque mejor que la búsqueda aleatoria, puede estancarse en óptimos locales sin mecanismos adicionales como reinicios o cruce ponderado.

5.4. Algoritmo Genético con Cruce Ponderado

5.4.1. Descripción

Esta versión introduce una mejora sobre el algoritmo genético básico al modificar el operador de cruce para que tenga en cuenta la calidad relativa de los padres. En lugar de combinar aleatoriamente los subconjuntos, el cruce ponderado asigna mayor peso al progenitor con mejor fitness, favoreciendo así la herencia de sus características más beneficiosas.

Además, en cada reproducción solo se selecciona el mejor hijo generado por la pareja de padres, descartando el otro. Esta estrategia reduce la generación de soluciones subóptimas, acelera la convergencia y mejora la estabilidad general del proceso evolutivo.

Se mantiene el uso de elitismo, selección por torneo y mutación con probabilidad fija, al igual que en la versión anterior.

5.4.2. Componentes del Algoritmo

A continuación, se describen los principales componentes añadidos o modificados para realizar esta versión con crude pronderado.

Cruce ponderado

El operador de cruce se basa en el fitness de los padres. Se asigna un mayor número de imágenes al progenitor con mejor rendimiento, lo que incrementa la probabilidad de heredar características positivas. Esta estrategia mejora la calidad de los hijos generados.

Mutación

Se mantiene la mutación aleatoria con probabilidad fija. Su propósito es introducir variabilidad en la población y evitar la convergencia prematura. En este caso, la mutación se aplica después del cruce ponderado.

Selección de hijo único

A diferencia de versiones anteriores, esta implementación evalúa los dos hijos generados y conserva únicamente el mejor. Esta selección más estricta reduce la propagación de soluciones poco prometedoras y acelera la convergencia del algoritmo.

5.4.3. Pseudocódigo

Para facilitar su lectura o replicación, se ha detallado el pseudocódigo del **Algorithm 4** Algoritmo Genético con Cruce Ponderado.

5.4.4. Aplicación en la Reducción de Datos

Este algoritmo permite una selección más refinada de subconjuntos de imágenes representativas al priorizar soluciones de alta calidad durante el cruce. Al favorecer a los progenitores con mejor rendimiento, se acelera la convergencia hacia subconjuntos efectivos, o que resulta en una reducción más precisa y estable del conjunto de entrenamiento.

5.4.5. Ventajas y Limitaciones

Ventajas:

Algorithm 4 Algoritmo Genético con Cruce Ponderado

```
Inicializar población y evaluar
while no se alcance el máximo de evaluaciones do
    Conservar el mejor individuo (elitismo)
    while población incompleta do
        Seleccionar padres por torneo
        Aplicar cruce ponderado
        Mutar hijos
        Evaluar ambos hijos
        Conservar solo el mejor hijo
    end while
    Reemplazar población
    Actualizar mejor si mejora
end while
return mejor individuo encontrado
```

- Favorece la herencia de características beneficiosas.
- Elimina soluciones débiles al conservar solo el mejor hijo.
- Acelera la convergencia sin sacrificar la calidad de la solución.

Limitaciones:

- Puede reducir la diversidad de la población si las soluciones convergen demasiado rápido.
- Requiere una evaluación adicional por cada pareja de padres (evaluar dos hijos).

5.5. Algoritmo Genético con Mutación Adaptativa

5.5.1. Descripción

Esta versión introduce una mejora centrada en el operador de mutación. En lugar de aplicar un número fijo de modificaciones, se implementa una **mutación adaptativa** que ajusta dinámicamente el número de intercambios en función del tamaño del subconjunto seleccionado.

Esta estrategia permite una exploración más equilibrada del espacio de soluciones, adaptándose mejor a las características de cada individuo. De

este modo, se evita tanto una mutación excesiva que degrade soluciones prometedoras como una mutación insuficiente que impida escapar de óptimos locales.

El resto de componentes del algoritmo se mantienen: selección por torneo, cruce ponderado, elitismo y evaluación del mejor hijo generado. La mutación adaptativa se aplica tras el cruce y antes de la evaluación.

5.5.2. Componentes del Algoritmo

Mutación adaptativa

El operador de mutación ajusta el número de intercambios según el tamaño relativo del subconjunto. Esto se calcula usando la siguiente fórmula:

$$\texttt{numswaps} = \text{mín}\left(\texttt{longitud} \times 0.15,\ \texttt{longitud} \times \texttt{ratio} \times 0.8\right)$$

donde `longitud` representa el número de imágenes disponibles y `ratio` el porcentaje seleccionado. Así, se consigue una mutación proporcional y contextual.

Preservación de la estructura

A pesar de la variabilidad introducida, se conserva la estructura binaria de selección, evitando cambios abruptos que rompan la proporción de clases o introduzcan ruido excesivo.

5.5.3. Pseudocódigo

Para facilitar su lectura o replicación, se ha detallado el pseudocódigo del **Algorithm 5** Algoritmo Genético con Mutación Adaptativa.

5.5.4. Aplicación en la Reducción de Datos

La mutación adaptativa mejora la capacidad del algoritmo para explorar el espacio de soluciones sin comprometer la calidad de los individuos. Esta versión demostró un mejor equilibrio entre exploración y explotación, especialmente en subconjuntos de tamaño variable o intermedio.

Algorithm 5 Algoritmo Genético con Mutación Adaptativa

```
Inicializar población y evaluar
while no se alcance el máximo de evaluaciones do
    Conservar el mejor individuo (elitismo)
    while población incompleta do
        Seleccionar padres por torneo
        Aplicar cruce ponderado
        Aplicar Mutación Adaptativa a los hijos
        Evaluar ambos hijos
        Conservar solo el mejor hijo
    end while
    Reemplazar población
    Actualizar mejor si mejora
end while
return mejor individuo encontrado
```

5.5.5. Ventajas y Limitaciones

Ventajas:

- Introduce una mutación más flexible y ajustada al contexto de cada individuo.
- Mejora la estabilidad de las soluciones frente a mutaciones agresivas.
- Favorece la diversidad genética sin perder precisión.

Limitaciones:

- Puede requerir calibración del rango de mutación para casos extremos.
- En subconjuntos muy pequeños, la variabilidad puede verse limitada.

5.6. Algoritmo Genético con Reinicio Poblacional

5.6.1. Descripción

Esta versión del algoritmo genético introduce una estrategia para evitar el estancamiento evolutivo mediante reinicios poblacionales. Cuando el segundo mejor individuo de la población no mejora durante dos generaciones consecutivas, se considera que el algoritmo ha dejado de progresar de forma significativa. En ese caso, se reinicia la población manteniendo únicamente el mejor individuo, y se genera el resto de forma aleatoria.

Este mecanismo permite escapar de óptimos locales y explorar nuevas regiones del espacio de soluciones sin perder por completo los avances logrados. El algoritmo conserva los operadores de selección, cruce ponderado y mutación, así como el elitismo.

5.6.2. Componentes del Algoritmo

Ahora, se describen los principales componentes añadidos o modificados para añadirle el reinicio poblacional al algoritmo genético con cruce ponderado.

Criterio de reinicio

El algoritmo monitoriza el rendimiento del segundo mejor individuo en cada generación. Si este no mejora en dos iteraciones consecutivas, se considera que la población ha entrado en un estado de estancamiento evolutivo y se procede a un reinicio.

Reinicio selectivo

Durante el reinicio, solo se conserva el mejor individuo de la población actual. El resto de la población se regenera completamente mediante una nueva inicialización aleatoria. Esto permite explorar nuevas regiones del espacio de soluciones sin perder los avances obtenidos.

Evaluación post-reinicio

Tras el reinicio, todos los nuevos individuos se evalúan y se reorganiza la población según su fitness. Se actualizan los valores del mejor y segundo mejor individuo, y se reanuda el ciclo evolutivo.

Persistencia de historial

Para no perder información relevante, se mantiene un historial acumulativo del fitness a lo largo de todos los reinicios. Esto permite realizar un análisis completo del proceso evolutivo una vez finalizada la ejecución.

5.6.3. Pseudocódigo

Para facilitar su lectura o replicación, se ha detallado el pseudocódigo del **Algorithm 6** Algoritmo Genético con Reinicio Poblacional.

Algorithm 6 Algoritmo Genético con Reinicio Poblacional

```
Inicializar población y evaluar
Guardar mejor y segundo mejor individuos
while no se alcance el máximo de evaluaciones do
    Aplicar elitismo
    while población incompleta do
        Seleccionar padres por torneo
        Aplicar cruce ponderado y mutación adaptativa
        Evaluar hijos y conservar el mejor
    end while
    if segundo mejor no mejora en 2 generaciones then
        Reiniciar población conservando solo el mejor
    end if
    Actualizar mejores si hay mejora
end while
return mejor solución
```

5.6.4. Aplicación en la Reducción de Datos

Esta estrategia mejora la robustez del algoritmo en entornos donde hay múltiples óptimos locales. Al reintroducir diversidad controlada, incrementa la probabilidad de encontrar subconjuntos más eficientes para el entrenamiento del modelo, especialmente en datasets complejos o con clases desbalanceadas.

5.6.5. Ventajas y Limitaciones

Ventajas:

- Permite escapar de óptimos locales sin descartar los avances previos.
- Favorece una exploración más amplia del espacio de soluciones.
- Mejora la estabilidad y robustez del algoritmo frente al estancamiento.

Limitaciones:

- Incrementa la complejidad de implementación y control del flujo.
- El reinicio puede introducir ruido si se activa de forma prematura.

5.7. Algoritmo Memético

El **algoritmo memético** es una extensión del enfoque genético tradicional que incorpora técnicas de búsqueda local dentro del proceso evolutivo. Su objetivo es combinar la exploración global del espacio de soluciones (propia de los algoritmos evolutivos) con la explotación local de buenas soluciones (propia de la optimización heurística). En este trabajo, se implementó una versión adaptada del algoritmo memético orientada a la selección de subconjuntos óptimos de imágenes para el entrenamiento de modelos convolucionales.

5.7.1. Descripción

El funcionamiento general del algoritmo memético se basa en una estructura genética clásica: generación de población inicial, selección por torneo, cruce, mutación y reemplazo elitista. Sin embargo, introduce una novedad clave: la aplicación probabilística de una búsqueda local sobre los individuos recién generados. Este procedimiento local intenta mejorar los hijos generados por cruce, evaluando soluciones vecinas mediante pequeñas modificaciones (mutaciones controladas).

5.7.2. Pseudocódigo

Para facilitar su lectura o replicación, se ha detallado el pseudocódigo del **Algorithm 7** Algoritmo Memético.

5.7.3. Aplicación en la reducción de datos

El **algoritmo memético** fue diseñado para encontrar subconjuntos de datos que optimicen métricas como la *accuracy* o el *F1-score* en un número reducido de evaluaciones. Su capacidad para ajustar localmente los subconjuntos permite afinar las selecciones iniciales generadas por los operadores evolutivos, lo que se traduce en soluciones de mayor calidad con menor dispersión.

5.7.4. Ventajas y limitaciones

Ventajas:

- Mejora puntual de soluciones mediante refinamiento local.
- Reduce la probabilidad de estancamiento en óptimos locales.

Algorithm 7 Algoritmo Memético.

La búsqueda local está limitada por el número total de evaluaciones permitidas y permite realizar mejoras incrementales únicamente cuando resultan beneficiosas.

```
1: Inicializar población y evaluar
2: Guardar mejor individuo
3: while evaluaciones < máximo do
4:     Aplicar elitismo
5:     while nueva población incompleta do
6:         Seleccionar padres por torneo
7:         Aplicar cruce ponderado
8:         Aplicar mutación
9:         if se cumple probabilidad de búsqueda local then
10:            Ejecutar búsqueda local sobre el hijo
11:            Evaluar vecinos y conservar el mejor
12:        else
13:            Evaluar hijo directamente
14:        end if
15:        Añadir hijo resultante a la nueva población
16:    end while
17:    Reemplazar población
18:    Actualizar mejor si mejora
19:    if hay estancamiento then
20:        Terminar
21:    end if
22: end while
23: return mejor solución
```

- Genera soluciones más consistentes y robustas frente a la aleatoriedad.

Limitaciones:

- Aumenta ligeramente el coste computacional por las evaluaciones adicionales de la búsqueda local.
- Requiere calibración adicional de parámetros como la probabilidad de búsqueda local o el tamaño del vecindario.

5.8. Versiones Libres de los Algoritmos Evolutivos

En esta sección se describen las versiones **libres** de algunos algoritmos evolutivos, caracterizadas por permitir que el número de imágenes seleccionadas evolucione dinámicamente a lo largo del proceso. A diferencia de las versiones fijas, donde se impone un porcentaje constante de selección, estas variantes adaptan el tamaño de los subconjuntos seleccionados mediante operadores ajustables y estocásticos. Esta propiedad resulta útil cuando no se conoce a priori la proporción óptima de instancias para entrenar un modelo eficiente.

5.8.1. Algoritmo Genético con Cruce Ponderado (Versión Libre)

La versión libre del Algoritmo Genético con Cruce Ponderado incorpora un operador de cruce flexible que ajusta automáticamente el tamaño de los subconjuntos generados. Al activar el parámetro `adjust_size`, el número de imágenes seleccionadas en cada hijo se calcula aleatoriamente dentro de un rango proporcional al tamaño de los padres, en este caso, entre el 50 % y el 150 % del tamaño base.

Esto permite al algoritmo explorar configuraciones más amplias o más compactas en busca del subconjunto óptimo. Además, el operador de mutación también se adapta para permitir fluctuaciones en la cantidad total de imágenes activas, aumentando así la diversidad estructural de la población.

5.8.2. Algoritmo Genético con Reinicio Poblacional (Versión Libre)

La versión libre del Algoritmo Genético con Reinicio Poblacional conserva la lógica de reinicio para evitar estancamientos, pero permite que cada nuevo individuo generado tenga una cantidad de imágenes seleccionadas distinta.

Este comportamiento se gestiona mediante el mismo mecanismo de cruce adaptativo que en la versión libre del genético v2.

Gracias a esta propiedad, cada reinicio poblacional no solo introduce nueva diversidad en el contenido de los subconjuntos, sino también en su escala. Esto es especialmente útil para escapar de regiones del espacio de soluciones con un tamaño fijo subóptimo.

5.8.3. Algoritmo Memético (Versión Libre)

La versión libre del Algoritmo Memético combina tanto la flexibilidad del cruce ponderado como la búsqueda local ajustable. Al igual que en los algoritmos anteriores, el tamaño de los subconjuntos puede variar en cada generación mediante un parámetro `adjust_size` activado tanto en los operadores evolutivos como en la heurística local.

Durante la búsqueda local, esta versión permite que un vecino no solo cambie el contenido de las imágenes seleccionadas, sino también su número total. Esta dualidad entre exploración global y explotación local con tamaño dinámico convierte al algoritmo en una herramienta poderosa para adaptarse a distintas complejidades del conjunto de datos, logrando soluciones robustas sin una configuración de selección fija.

Capítulo 6

Implementación

En este capítulo se presenta en detalle la arquitectura técnica del sistema implementado, incluyendo los componentes y módulos principales, las herramientas específicas empleadas en la construcción del sistema, y los elementos clave para optimizar el rendimiento de los algoritmos y su evaluación.

6.1. Descripción del Sistema

La estructura del proyecto se organizó modularmente para facilitar el acceso, el mantenimiento y la extensibilidad del código fuente. La organización de carpetas es la siguiente:

- `data` – Conjunto de datos utilizados en los experimentos.
- `docs` – Documentación del proyecto en latex.
  - `bibliografia` – Archivos relacionados con las referencias bibliográficas.
  - `capitulos` – Archivos individuales para cada capítulo del documento.
  - `config` – Archivos de configuraciones de la documentación LaTeX.
  - `imagenes` – Imágenes utilizadas en la documentación.
  - `out` – Archivos generados por el compilador de LaTeX.
  - `portada` – Archivo portada del documento.
  - `prefacio` – Archivo prefacio del documento.
  - `proyecto.tex` – Archivo principal de LaTeX que compila el documento.

- `img` – Imágenes generadas automáticamente durante los experimentos.
- `LICENSE` – Términos de distribución del proyecto.
- `logs` – Registros de las ejecuciones, incluyendo tiempos de inicio, fin y resultados intermedios de los algoritmos.
- `README.md` – Descripción general.
- `requirements.txt` – Dependencias del proyecto.
- `results` – Resultados de los experimentos.
  - `csvs` – Resultados de las ejecuciones guardados en tablas.
  - `salidas` – Salidas en bruto de consola.
- `scripts` – Scripts de ejecución automática, comparación de experimentos y generación de gráficos finales.
- `src` – Código fuente principal del proyecto.
  - `algorithms` – Implementaciones de los algoritmos.
  - `main.py` – Módulo principal de ejecución individual.
- `tmp` – Ficheros temporales generados durante la ejecución.
- `utils` – Módulos de apoyo, como clases auxiliares, generación de gráficos y funciones utilitarias.

6.2. Herramientas y Lenguajes de Programación

El desarrollo del proyecto se ha llevado a cabo utilizando **Python 3.10** [24] como lenguaje principal, debido a su versatilidad y amplia adopción en el campo del **aprendizaje profundo** y la **manipulación de datos**. Python es conocido por su facilidad de uso, extensibilidad y la gran cantidad de bibliotecas disponibles para el procesamiento de datos y la implementación de modelos de **machine learning**.

Las principales bibliotecas empleadas durante el desarrollo son las siguientes:

- **PyTorch 2.3.1** [25, 26]: Para la construcción, entrenamiento y optimización de modelos de aprendizaje profundo. PyTorch fue elegido por su flexibilidad y capacidad para ejecutarse eficientemente en GPU.
- **Scikit-learn 1.5.2** [24]: Para la evaluación de los modelos se utilizaron métricas estándar [27]. Su API permite una integración fluida con PyTorch y otros módulos.

- **Numpy 2.0.0** [28]: Para operaciones matemáticas y manipulación de matrices, siendo una herramienta esencial en el procesamiento de datos.
- **Polars 1.9.0** [29]: Biblioteca para manejar DataFrames de gran tamaño, elegida por su rendimiento superior en comparación con Pandas.
- **Matplotlib 3.9.2** [30]: Biblioteca utilizada para la generación y visualización de gráficas.
- **Seaborn 0.13.2** [31]: Estilización avanzada de gráficos estadísticos.
- **Openpyxl 3.1.5** [32]: Generación automática de archivos Excel a partir de resultados experimentales.

Cada una de estas herramientas fue seleccionada por su robustez y su idoneidad para cumplir con los requisitos específicos del proyecto, facilitando tanto la implementación de los algoritmos meméticos como la reducción y el análisis de los datos utilizados en los modelos de aprendizaje profundo.

6.3. Gestión de Dependencias

Para garantizar que el proyecto se ejecute correctamente y todas las bibliotecas necesarias estén disponibles, se ha utilizado un archivo `requirements.txt`. Este archivo contiene una lista de todas las bibliotecas y sus versiones específicas que el proyecto requiere.

Para el **desarrollo local**, se ha optado por crear un entorno virtual utilizando `venv` [33]. Esta práctica permite aislar las dependencias del proyecto de otros proyectos en la máquina, evitando conflictos entre versiones de bibliotecas.

Para la **implementación en el servidor**, se ha utilizado `conda` [34] como gestor de paquetes y entornos. Conda facilita la gestión de entornos y la instalación de bibliotecas, especialmente en configuraciones más complejas.

Esto facilita la reproducibilidad del proyecto y minimiza posibles conflictos de versión, lo que es fundamental para mantener la integridad del código y el rendimiento de las aplicaciones.

6.4. Arquitectura de la Implementación

La arquitectura de la implementación se organiza en varios módulos, que a continuación se describen en detalle:

6.4.1. Módulo de Algoritmos

Ubicado en `src/algorithms/` este módulo contiene las implementaciones principales de los algoritmos desarrollados en el proyecto.

Este módulo utiliza la arquitectura GPU para maximizar la velocidad de ejecución y está diseñado para ser escalable, permitiendo la inclusión de nuevos operadores meméticos si es necesario.

6.4.2. Núcleo de Ejecución

El módulo `main.py` centraliza la ejecución de un experimento individual, inicializando configuraciones, entrenando el modelo y generando gráficos.

Los pasos de la función principal de `main.py` es:

1. **Establece Configuración Inicial**: Configura una semilla, elige el dataset y prepara un archivo de log.
2. **Inicia el Proceso del Algoritmo**: Según el nombre del algoritmo (algoritmo) especificado, se llama a la función correspondiente (por ejemplo, genetic_algorithm, memetic_algorithm, etc.).
3. **Almacena Resultados**: Una vez que el algoritmo termina, registra la duración, los resultados y la métrica final en un archivo.
4. **Visualiza Resultados**: Si hay datos de fitness, genera una gráfica de la evolución del fitness a lo largo del proceso.
5. **Genera un Resumen**: Calcula estadísticas adicionales (como porcentaje de clases seleccionadas en Paper, Rock y Scissors), y devuelve estos resultados junto con el historial de fitness.

Adicionalmente, los scripts `generator.py` y `generator_initial.py` permiten automatizar experimentos masivos combinando distintos algoritmos, porcentajes iniciales y modelos de red neuronal.

6.4.3. Módulo de Utilidades

La carpeta `utils/` contiene funciones auxiliares:

- **utils_plot.py**: Generación de gráficos.
- **classes.py**: Definición de enumeraciones para algoritmos, métricas, datasets y modelos.

- **utils.py**: Funciones de ayuda como el cálculo de métricas o la creación de diccionarios de selección de imágenes.

Estos módulos se encargan de generar gráficas comparativas entre distintos porcentajes o algoritmos y en generar un CSV con los datos finales para ser analizados.

6.4.4. Scripts de Ejecución en GPU

En scripts, se encuentran los programas necesarios para ejecutar los algoritmos en un servidor GPU, lo que permite maximizar la eficiencia en el entrenamiento y la evaluación de modelos.

1. **Configuración de GPU**: Los scripts están configurados para identificar y utilizar las GPU disponibles en el servidor, reduciendo los tiempos de entrenamiento de modelos.

2. **Optimización de Ejecución**: Se implementaron configuraciones de batch size y técnicas de procesamiento paralelo en PyTorch, aprovechando la memoria y el poder de procesamiento de las GPU.

Estos scripts están diseñados para ser ejecutados en un entorno de servidor, reduciendo los tiempos de prueba en el entorno local y permitiendo un análisis iterativo más rápido.

6.5. Consideraciones de Optimización

Durante el desarrollo, se optimizaron varios aspectos para mejorar el rendimiento del sistema:

1. **Aceleración en GPU**: Todas las operaciones de cálculo intensivo fueron migradas a la GPU mediante PyTorch.

2. **Uso Eficiente de Memoria**: Con Polars y Numpy, se optimizó el manejo de grandes volúmenes de datos, utilizando tipos de datos específicos para reducir el uso de memoria.

3. **Automatización de Evaluaciones**: Las pruebas de rendimiento se automatizaron, permitiendo una evaluación continua sin intervención manual.

4. **Control de reproducibilidad**: Se fijaron semillas aleatorias en todas las librerías involucradas (random, numpy, torch, cuda) y se desactivaron los algoritmos no deterministas de cuDNN [35]. Esta medida garantiza que las ejecuciones del sistema produzcan resultados consistentes entre sesiones, algo esencial en entornos de evaluación comparativa.

5. **Diagnóstico automático de GPU**: Implementación de un script (cuda-diagnostic.py) que comprueba disponibilidad de CUDA y dispositivos antes de lanzar experimentos, garantizando un entorno correcto.

Además, se implementó un mecanismo de **early stopping** basado en la ausencia de mejora del valor de fitness durante un número determinado de evaluaciones consecutivas. Aunque no se utiliza una pérdida de validación explícita como en enfoques tradicionales, este enfoque funcionalmente cumple el mismo propósito: detener el algoritmo cuando se detecta estancamiento, reduciendo así el coste computacional innecesario [36].

Gracias a estas optimizaciones, el sistema permite explorar un amplio abanico de configuraciones de manera eficiente, manteniendo la robustez y estabilidad de los resultados.

6.6. Métricas de Evaluación

Para evaluar el rendimiento de los modelos de clasificación utilizados en los experimentos, se han empleado cuatro métricas estándar en el ámbito del aprendizaje automático: **accuracy**, **precisión**, **recall** y **F1-score**. Dado que se trata de un problema multiclase, estas métricas se han calculado utilizando el promedio *macro*, lo que implica computarlas individualmente para cada clase y luego obtener la media aritmética.

A continuación se describen y formalizan cada una de estas métricas:

- **Accuracy** (exactitud): representa la proporción de predicciones correctas sobre el total de ejemplos.

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

En el caso multiclase, se generaliza como:

$$\text{Accuracy} = \frac{\text{Número de predicciones correctas}}{\text{Total de predicciones}}$$

- **Precisión**: mide cuántas de las instancias clasificadas como positivas fueron realmente positivas. En el caso multiclase (macro), se calcula como:

$$\text{Precisión}_{\text{macro}} = \frac{1}{C} \sum_{i=1}^{C} \frac{TP_i}{TP_i + FP_i}$$

- **Recall** (exhaustividad): indica cuántas de las instancias realmente positivas fueron correctamente identificadas por el modelo:

$$\text{Recall}_{\text{macro}} = \frac{1}{C} \sum_{i=1}^{C} \frac{TP_i}{TP_i + FN_i}$$

- **F1-score**: es la media armónica entre precisión y recall, útil cuando se desea un equilibrio entre ambas:

$$\text{F1-score}_{\text{macro}} = \frac{1}{C} \sum_{i=1}^{C} \frac{2 \cdot \text{Precisión}_i \cdot \text{Recall}_i}{\text{Precisión}_i + \text{Recall}_i}$$

Donde:

- TP_i: verdaderos positivos de la clase i
- FP_i: falsos positivos de la clase i
- FN_i: falsos negativos de la clase i
- C: número total de clases

Para facilitar la comprensión de estos conceptos, en la Figura 6.1 se muestra una representación gráfica de una matriz de confusión.

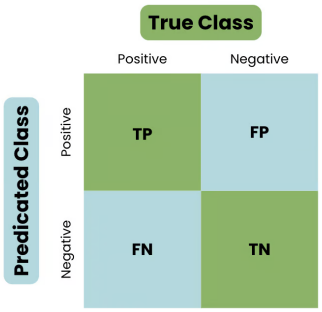

Figura 6.1: Representación gráfica de una matriz de confusión

Estas métricas fueron implementadas utilizando la librería `scikit-learn`, que permite su cálculo eficiente a partir de las predicciones del modelo y las etiquetas reales del conjunto de validación.

Capítulo 7

Entorno de Pruebas

En este capítiulo se describe el entorno de pruebas utilizado para evaluar las pruebas realizadas. Además, se detallan los datasets empleados, el diseño de los experimentos y el procedimiento de ejecución y evaluación, junto a como se expresan y muestran los resultados obtenidos.

7.1. Datasets utilizados

En el aprendizaje profundo, los datasets son colecciones de datos etiquetados o no etiquetados que se utilizan para entrenar modelos. Estos conjuntos de datos contienen ejemplos organizados que representan la entrada para el modelo y, en muchos casos, también las etiquetas correspondientes que indican la salida deseada. Los datasets varían en tamaño, calidad y tipo, dependiendo de la tarea a resolver, como la clasificación de imágenes, el reconocimiento de patrones o la predicción de series temporales.

A continuación, se van a explicar cada uno de los Datasets que se han utilizado en el desarrollo del proyecto.

7.1.1. Rock, Paper, Scissors (Piedra, Papel, Tijera)

Rock, Paper, Scissors [37] es un conjunto de datos creado por Laurence Moroney que se utiliza para la clasificación de imágenes de manos representando los gestos de 'piedra', 'papel' y 'tijeras'.

En la figura 7.1 se han mostrado una imagen de cada una de las clases del dataset Rock, Paper, Scissors, para que se pueda observar la similitud entre las distintas clases.

Rock Paper Scissors

Figura 7.1: Ejemplos de imágenes del dataset Rock, Paper, Scissors

Estructura del Dataset

El conjunto de datos contiene aproximadamente 2,500 imágenes, distribuidas en tres categorías: piedra, papel y tijeras. Las imágenes están en color y tienen un tamaño de 300x300 píxeles.

Como se puede observar en la Figura 7.2, las imágenes están organizadas en directorios según su función en el entrenamiento (entrenamiento, validación o prueba) y, dentro de cada partición, se dividen a su vez por clases del dataset.

Formato de los Datos

Las imágenes están en formato JPEG (`.jpg`). Para su procesamiento, se han aplicado técnicas de preprocesamiento adaptadas a los requerimientos del modelo.

Uso del Dataset

Este dataset se ha utilizado para evaluar el rendimiento del modelo en un problema de clasificación de imágenes con múltiples clases, pero siendo un dataset sencillo y con un número de clases pequeño. Además, permite explorar la eficacia de los algoritmos meméticos en un entorno más cercano al reconocimiento de objetos.

Correcciones en la División de Datos

Según la nota observada en el README del dataset:

> *Note: in the source, Laurence calls "validation" as the "test", and "test" the "validation".*

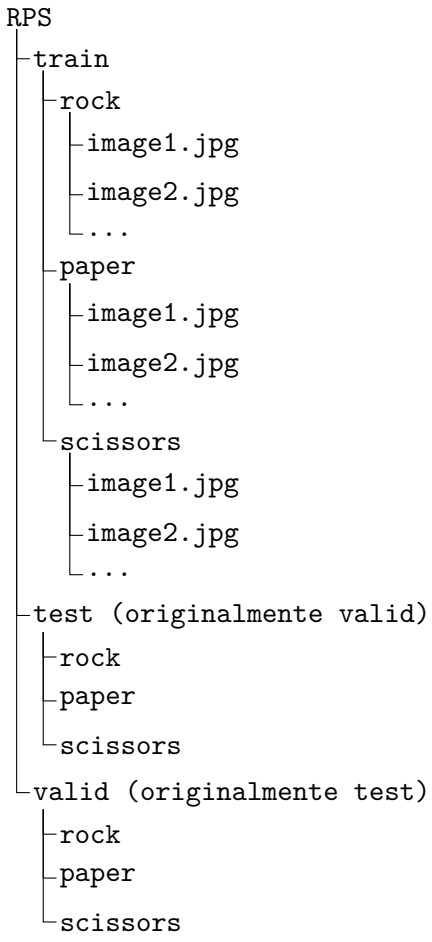

Figura 7.2: Estructura de carpetas del dataset Rock, Paper, Scissors

se han renombrado las particiones de `test` y `valid` para que correspondan correctamente con sus propósitos.

Licencia y uso

Este conjunto de datos se distribuye bajo la licencia **Creative Commons Attribution 4.0 International (CC BY 4.0)**, lo que permite su uso, modificación y distribución con la condición de otorgar el crédito adecuado a los creadores originales [**moroneyLaurenceMoroneyAI**].

7.1.2. PAINTING (Art Images: Drawing/Painting/Sculptures/Engravings)

El dataset **Art Images: Drawing/Painting/Sculptures/Engravings** es una colección de aproximadamente 9,000 imágenes organizadas en cinco categorías de arte: dibujos, pinturas, esculturas, grabados y arte iconográfico.

Drawings Painting Iconography

Engraving Sculpture

Figura 7.3: Ejemplos de clases en el dataset PAINTING

En la figura 7.3 se han mostrado una imagen de cada una de las clases del dataset PAINTING, para que se pueda observar la variabilidad de las imágenes.

Estructura del Dataset

Como se puede observar en la Figura 7.4, las imágenes están organizadas en directorios según su categoría artística, que en este caso corresponden a las distintas clases del dataset, previamente divididas en conjuntos de entrenamiento y prueba.

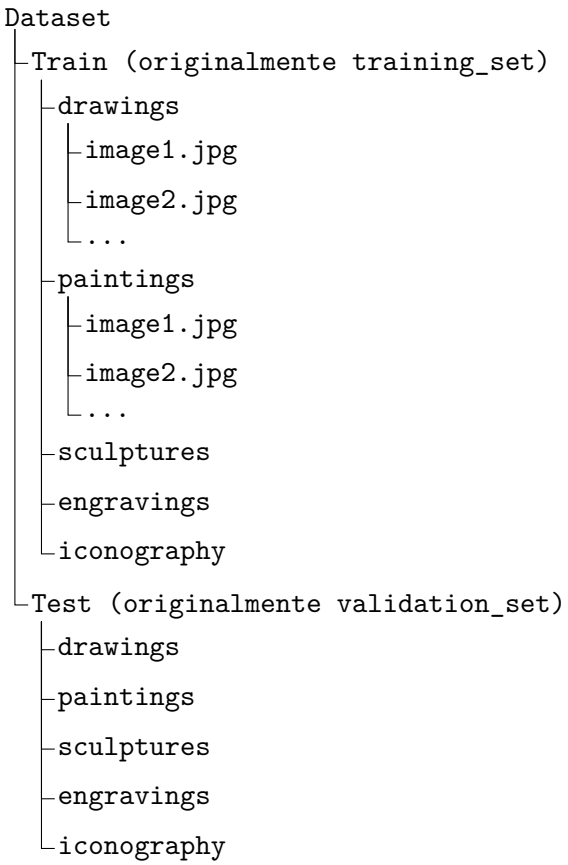

Figura 7.4: Estructura de carpetas del dataset `PAINTING`

Formato de los Datos

Todas las imágenes están en formato JPEG (`.jpg`) y presentan variaciones en resolución y dimensiones. Se han aplicado técnicas de preprocesamiento para homogenizar las características de las imágenes.

Uso del Dataset

Este dataset se ha utilizado para entrenar y evaluar modelos de clasificación de imágenes en un entorno diferente al RPS. Con este dataset, se ha comprobado el funcionamiento para evaluar los algoritmos con un dataset un poco mas complejo que el RPS, con un par de clases más y con un número mayor de imágenes.

Correcciones en la División de Datos

Observando los tamaños de la división de los datos, y teniendo en cuenta que la divisón de los datos suele ser en train y test, se ha decidido por renombrar las particiones de `valid` por `test` para que corresponda correctamente con su propósito. Y el set de validation lo he obtenido separando el set de train, normalmente haciendo una división 80 % test y 20 % valid.

Acceso al Dataset

Inicialmente, el dataset se descargó desde Kaggle [38]

Sin embargo, debido a la presencia de archivos innecesarios y algunas imágenes corruptas, se optó por una versión limpia disponible en Kaggle [39].

Licencia y Uso

Antes de su uso, se revisaron los términos y condiciones establecidos en la página de Kaggle para asegurar el cumplimiento con las licencias y restricciones aplicables.

7.1.3. Comparación entre datasets

Gracias a la tabla 7.1 que con las características más relevantes de los datasets utilizados, permite entender mejor la complejidad relativa de cada conjunto y cómo pueden influir en el comportamiento de los algoritmos:

Dataset	Nº Imágenes	Nº Clases	Formato	Tamaño Imagen
Rock, Paper, Scissors	~2.500	3	JPG	300×300 px
PAINTING	~9.000	5	JPG	Variable

Tabla 7.1: Resumen comparativo de los datasets utilizados

La selección de estos dos datasets responde a la necesidad de evaluar los algoritmos meméticos en distintos niveles de complejidad. El dataset **Rock, Paper, Scissors** se utilizó en las primeras fases del proyecto como punto de partida, ya que ofrecía un entorno sencillo y controlado, con un número reducido de clases y una estructura equilibrada. Esto permitió desarrollar las bases del sistema y probar las primeras versiones de los algoritmos de manera más ágil y con menor complejidad computacional.

Por su parte, el dataset **PAINTING** se empleó posteriormente para validar el comportamiento de los algoritmos en un entorno más exigente. Al incluir cinco categorías de arte con distintos estilos visuales, este conjunto

introdujo una mayor variabilidad tanto semántica como estructural, lo que permitió evaluar la robustez y capacidad de generalización de las soluciones desarrolladas.

7.2. Diseño de los experimentos

La fase experimental se organizó en varias etapas. Inicialmente se optó por un dataset simple (*Rock, Paper, Scissors*) para validar el funcionamiento general del sistema. Posteriormente, se realizaron pruebas con datasets más exigentes. Los experimentos se repitieron utilizando diferentes porcentajes iniciales de datos (10 %, 25 %, 50 % y 75 %) para estudiar cómo afecta la cantidad de datos seleccionados al rendimiento del modelo.

Con el fin de asegurar la consistencia entre ejecuciones experimentales, se aplicaron las medidas de control de reproducibilidad descritas en el Apartado 6.5. Esto permitió comparar algoritmos en condiciones homogéneas, evitando variaciones indeseadas causadas por componentes aleatorios del entorno de ejecución.

En cada prueba, se realizaron 5 ejecuciones en paralelo, cada una utilizando una semilla distinta. Esta estrategia permitió obtener resultados promedio más robustos frente a la aleatoriedad del proceso evolutivo, asegurando una mayor fiabilidad estadística.

Los apartados siguientes explican con mayor detalle el procedimiento adoptado para llevar a cabo dichas ejecuciones.

7.3. Procedimiento de Ejecución y Evaluación

Para garantizar la consistencia y objetividad en la comparación entre algoritmos, se diseñó un procedimiento experimental sistemático y replicable. Cada ejecución se realizó bajo las mismas condiciones computacionales y utilizando los mismos parámetros base, salvo en aquellos casos en que se deseaba estudiar una variación concreta, como los distintos porcentajes iniciales o el uso de metaheurísticas con porcentajes libres.

7.3.1. Métricas de Evaluación

Para evaluar el rendimiento de los modelos se utilizaron métricas estándar como **accuracy**, **precisión**, **recall** y **F1-score**, calculadas sobre el conjunto de validación tras cada evaluación. Para una definición formal de estas métricas, véase el Apartado 6.6.

Estas métricas fueron calculadas utilizando las funciones de `scikit-learn`, a partir de las predicciones del modelo y las etiquetas reales correspondientes a los subconjuntos de imágenes seleccionados por cada algoritmo.

7.3.2. Evaluaciones por Ejecución

Se buscó un equilibrio entre la cantidad de evaluaciones y el tiempo de ejecución, permitiendo una exploración suficiente del espacio de soluciones sin comprometer la eficiencia computacional. Por ello, cada algoritmo fue configurado para realizar un máximo de 100 evaluaciones por ejecución, independientemente del tipo de algoritmo utilizado, con el fin de mantener la equidad comparativa.

Cada evaluación consistía en generar un subconjunto de datos, entrenar el modelo correspondiente (ResNet50 o MobileNetV2), y calcular su *fitness* de acuerdo con las métricas mencionadas.

El número de evaluaciones sin mejora también fue monitorizado para aplicar criterios de parada anticipada, explicados previamente en el Apartado 6.5, reduciendo así el tiempo computacional en caso de estancamiento.

Visualización de resultados

Para facilitar la comparación entre algoritmos, a lo largo de este capítulo se incluyen representaciones gráficas de los resultados obtenidos mediante **boxplots** y **barplots**. Ambos tipos de gráficos permiten visualizar el comportamiento global de cada algoritmo a partir de múltiples ejecuciones con distintas semillas.

Los boxplots (diagramas de caja) muestran la distribución estadística de los valores obtenidos. En la Figura 7.5 se presenta un ejemplo anotado que ilustra las distintas partes de este tipo de gráfico. La línea central de la caja representa la **mediana**, mientras que los bordes inferior y superior corresponden al **primer cuartil** (Q1) y **tercer cuartil** (Q3), respectivamente. La diferencia entre ellos define el **rango intercuartílico** (IQR), que contiene el 50 % central de los valores. Las líneas que se extienden desde la caja (conocidas como *bigotes*) alcanzan típicamente hasta 1.5 veces el IQR. Los puntos que quedan fuera de ese rango se consideran **valores atípicos**, lo que permite detectar ejecuciones excepcionales. Esta representación es especialmente útil para comparar la tendencia central, la dispersión y la estabilidad de los resultados obtenidos por los distintos algoritmos.

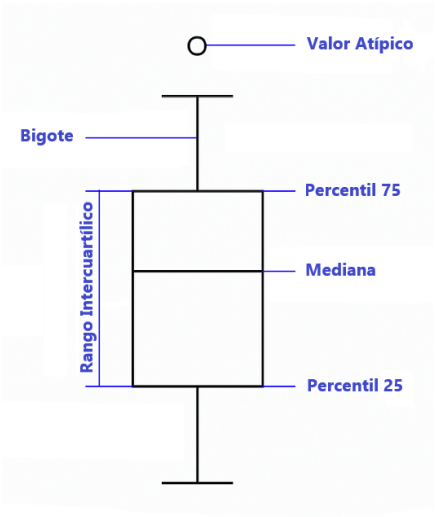

Figura 7.5: Ejemplo de boxplot explicado.

Los barplots (gráficos de barras), por su parte, se emplean para representar valores agregados como medias o proporciones, y son útiles para observar cómo varía una métrica concreta entre distintos algoritmos, modelos o configuraciones.

7.3.3. Repeticiones y Semillas

Con el objetivo de obtener resultados estadísticamente significativos y reducir el efecto de la aleatoriedad, cada configuración experimental fue ejecutada 5 veces, utilizando 5 semillas distintas. Los resultados presentados en las tablas y gráficos corresponden a la media de esas ejecuciones, junto con medidas de dispersión cuando procede, como los boxplots.

Cabe destacar que, en el caso de los boxplots, en lugar de representar la media de las 5 ejecuciones por configuración, se optó por incluir todos los valores individuales obtenidos con las distintas semillas. Esta decisión permite visualizar una distribución más realista del comportamiento de cada algoritmo, resaltando mejor la mediana, así como los valores máximos y mínimos alcanzados durante las ejecuciones.

7.3.4. Tiempos de Ejecución

Cada evaluación implicaba entrenar un modelo desde cero, por lo que los tiempos de ejecución fueron considerables. Por ejemplo, una ejecución completa con 100 evaluaciones podía tardar entre 30 minutos y 2 horas, dependiendo del algoritmo y del modelo utilizado.

Los algoritmos más complejos, como el memético o las versiones con reinicio poblacional, requerían un mayor tiempo de ejecución debido a las operaciones adicionales de mejora local o regeneración de población.

Capítulo 8

Resultados y Análisis

En este capítulo se exponen los experimentos realizados y los resultados obtenidos en los distintos escenarios evaluados. El objetivo principal fue analizar el rendimiento de los modelos entrenados con conjuntos de datos reducidos, seleccionados mediante algoritmos meméticos y evolutivos.

8.1. Evaluación con el conjunto completo

Como punto de partida, se evaluó el rendimiento de los modelos convolucionales entrenados con el 100 % del conjunto de datos. Esta prueba sirve como referencia para contrastar los resultados obtenidos mediante las técnicas de reducción aplicadas posteriormente. Se utilizó tanto ResNet50 como MobileNetV2, y se midieron métricas como accuracy, precisión, recall y F1-score sobre el conjunto de validación.

Porcentaje Inicial	Duración	Accuracy (Avg)	Precision (Avg)	Recall (Avg)	F1-score (Avg)	Evaluaciones Realizadas
100	00:02:42	87,90 %	88,96 %	87,90 %	87,81 %	1

Tabla 8.1: Resultados de ResNet50 entrenado con el 100 % del conjunto de datos.

Los resultados en la tabla 8.1 obtenidos representan el rendimiento máximo alcanzable en condiciones ideales, sin reducción de datos, lo que permite establecer un techo de rendimiento frente al cual comparar las demás técnicas.

8.2. Evaluación del enfoque aleatorio

El segundo experimento consistió en aplicar una selección aleatoria de ejemplos para distintos porcentajes iniciales de datos (10 %, 25 %, 50 % y 75 %). Este enfoque, descrito en el Apartado 5.1, sirvió como línea base para evaluar la eficacia de los enfoques metaheurísticos.

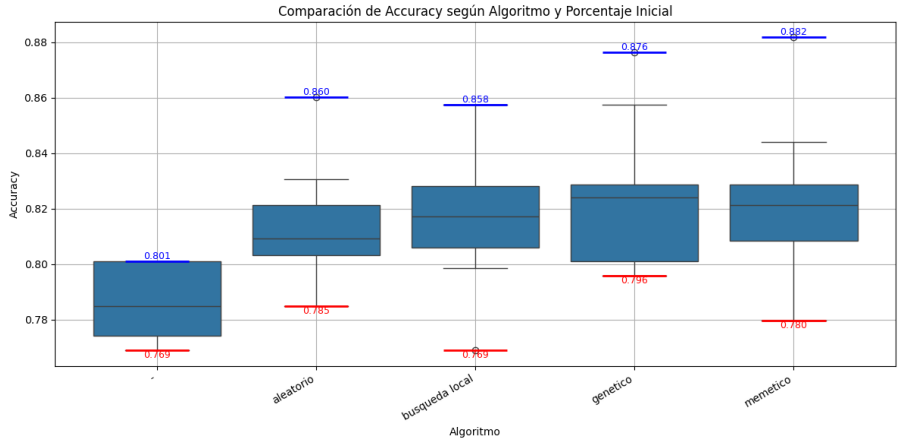

Figura 8.1: Boxplot comparando el *accuracy* alcanzado por cada algoritmo de los iniciales.

Falta modificar la tabla 8.1 para que compare el 100 % con el aleatorio de resnet y ajustar a

Falta añadir tabla comparando resultados por porcentajes iniciales.

Tal como se observa en el boxplot generado (Figura 8.1), como era de esperar, presenta una alta dispersión, pero el rendimiento es mejor que el resultado con el 100 % del dataset. Ademas, en esta otra tabla [tabla comparando resultados por porcentajes iniciales] los resultados mejoran de forma proporcional al incremento del porcentaje de imágenes seleccionadas. Este comportamiento se justifica por la ausencia de una estrategia que guíe la selección de datos, lo que da lugar a conjuntos de entrenamiento inconsistentes y resalta la necesidad de aplicar técnicas más sofisticadas para obtener resultados consistentes con menores volúmenes de datos.

8.3. Comparación entre modelos convolucionales

Antes de analizar los algoritmos de reducción, se comparó el comportamiento de los modelos **ResNet50** y **MobileNetV2** bajo las mismas condiciones de entrenamiento y subconjuntos aleatorios.

Los resultados de la tabla 8.2 muestran que ResNet50 mostró consisten-

Algoritmo	Porcentaje Inicial	Duración	Accuracy (Avg)	Precision (Avg)	Recall (Avg)	F1-score (Avg)	Evaluaciones Realizadas
Modelo ResNet50							
aleatorio	10	00:45:08	76,55 %	81,80 %	76,55 %	76,25 %	100
aleatorio	20	01:10:27	81,77 %	84,70 %	81,77 %	81,59 %	100
aleatorio	50	02:24:49	87,14 %	88,09 %	87,14 %	86,97 %	100
aleatorio	100	00:02:42	87,90 %	88,96 %	87,90 %	87,81 %	1
Modelo MobileNet							
aleatorio	10 %	00:29:29	72,31 %	76,40 %	72,31 %	69,62 %	100
aleatorio	20 %	00:50:36	76,48 %	78,82 %	76,48 %	75,58 %	100
aleatorio	50 %	01:54:09	75,56 %	79,72 %	75,56 %	74,67 %	100
-	100 %	00:03:16	78,60 %	81,55 %	78,60 %	77,68 %	1

Tabla 8.2: Comparativa de resultados de la generación inicial utilizando el algoritmo **aleatorio** con los modelos **ResNet50** y **MobileNet**.

temente mejores métricas, aunque a costa de mayores tiempos de entrenamiento. Por su parte, MobileNetV2 demostró ser más eficiente, resultando útil para entornos con restricciones computacionales.

Esta comparación justificó la elección de MobileNet como modelo principal en el resto de los experimentos, dada su superior eficiencia en términos de tiempo y recursos, posibilitando la facilidad de realizar múltiples pruebas en un tiempo razonable.

8.4. Estudio de la búsqueda local

Como se describió en el apartado correspondiente (ver Sección 5.2), la búsqueda local permite mejorar progresivamente una solución inicial mediante pequeñas modificaciones guiadas por el rendimiento. En este apartado se evalúa su efectividad como alternativa más estructurada frente al enfoque aleatorio, pero sin llegar a la complejidad de los algoritmos evolutivos. Su inclusión busca analizar hasta qué punto una estrategia simple pero guiada puede generar subconjuntos de datos más representativos y consistentes.

En la Figura 8.2 se muestran los resultados mediante un boxplot que permite observar la distribución completa de valores obtenidos en las distintas ejecuciones.

Se puede apreciar que el algoritmo de búsqueda local mejora claramente la mediana del *accuracy* respecto al enfoque aleatorio. Mientras que el enfoque aleatorio se sitúa en torno a una mediana de 0.787, la búsqueda local alcanza una mediana superior, próxima a 0.818. Esta diferencia refleja una mayor capacidad del algoritmo local para generar subconjuntos más representativos y eficaces.

Además, los valores máximos alcanzados por ambos algoritmos son simi-

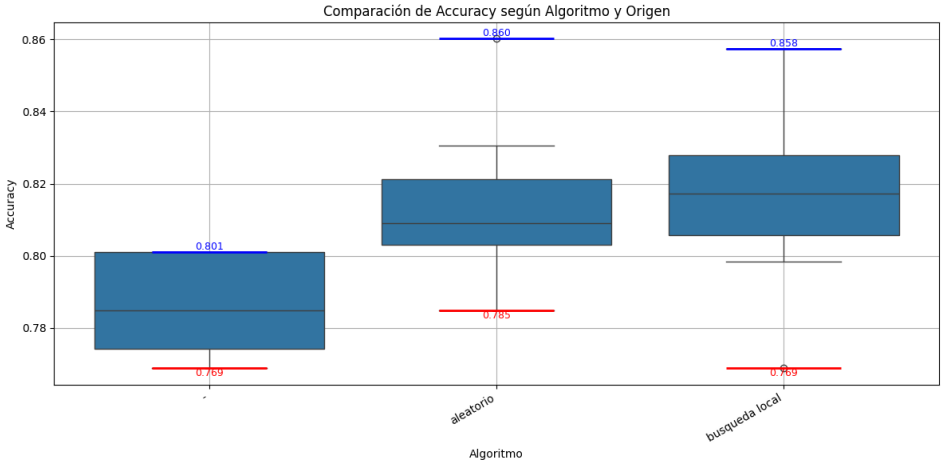

Figura 8.2: Boxplot comparando el algoritmo aleatorio con la búsqueda local usando *accuracy*.

lares (en torno a 0.858-0.860), pero la búsqueda local muestra una dispersión más acotada hacia valores altos, lo que sugiere mayor estabilidad en sus resultados. En cambio, el algoritmo aleatorio presenta una mayor dispersión hacia valores bajos y una mayor sensibilidad a la aleatoriedad de las selecciones, como se evidencia en su menor valor mínimo (0.769 frente a 0.785 en búsqueda local).

Esto pone de manifiesto que, aunque el algoritmo aleatorio puede ocasionalmente alcanzar buenos resultados, la búsqueda local ofrece una mejor consistencia y fiabilidad, con menos varianza entre ejecuciones y una tendencia general a obtener subconjuntos de entrenamiento más efectivos.

8.5. Estudiando la mejora metaheurística

Con el objetivo de superar las limitaciones observadas (como el riesgo de estancamiento o la exploración poco estructurada del espacio de soluciones) se incorporó un enfoque evolutivo más completo: el algoritmo genético, descrito en la Sección 5.3, el cual sirvió como punto de partida para explorar la aplicación de estrategias metaheurísticas en la selección de subconjuntos representativos de imágenes. Su estructura evolutiva, basada en selección por torneo, cruce e incorporación de mutación, ofrecía ya desde sus primeras versiones una capacidad superior para generalizar, en comparación con métodos más simples como la búsqueda local.

Tal como se observa en la Figura 8.3, el algoritmo genético consigue una **mediana de *accuracy*** más alta que la búsqueda local, reflejando

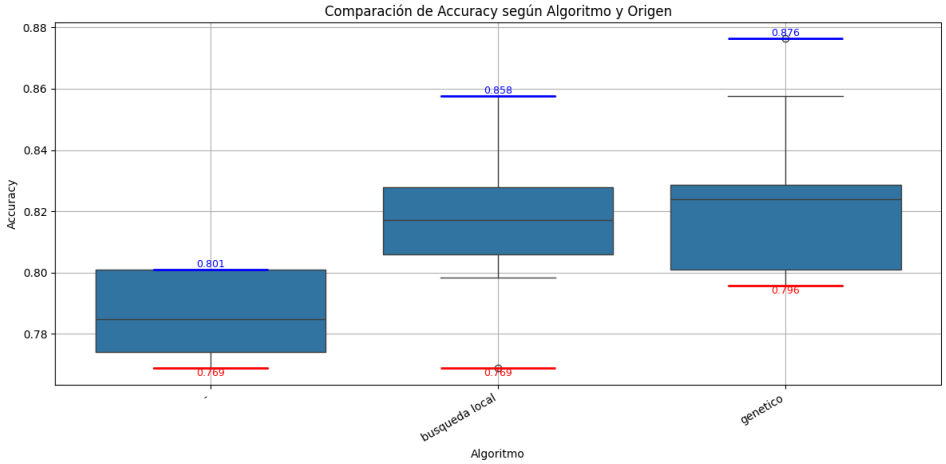

Figura 8.3: Boxplot comparando la búsqueda local con el algoritmo genético v1 usando *accuracy*.

un rendimiento medio más consistente. Además, presenta un valor máximo superior (alcanza hasta **0.876**), lo que evidencia su mayor potencial para encontrar soluciones de alta calidad.

No obstante, también se aprecia una ligera mayor dispersión en los resultados del algoritmo genético, particularmente hacia los valores bajos. Esto indica que, pese a su capacidad exploratoria, el genético puede generar soluciones poco efectivas si no se controlan adecuadamente ciertos operadores como el cruce o la mutación. De hecho, su valor mínimo (**0.796**) es superior al de la búsqueda local en esta comparativa, pero deja margen para mejoras en la presión selectiva o en mecanismos que eviten estancamientos.

¿Sobra el siguiente apartado? La búsqueda local, por su parte, mantiene un comportamiento más estable, aunque con una mediana ligeramente inferior. Su distribución es más concentrada y limitada en el extremo superior, lo que evidencia su carácter más explotador pero con menor capacidad para alcanzar soluciones óptimas globales.

Estos resultados sirvieron como evidencia empírica para continuar desarrollando nuevas versiones del algoritmo genético, incorporando mejoras específicas en sus operadores con el fin de aprovechar su capacidad exploratoria y, al mismo tiempo, mitigar sus limitaciones.

8.6. Modificando el operador de cruce

La primera mejora introducida al algoritmo genético consistió en reemplazar el cruce aleatorio por un cruce ponderado, donde se prioriza la contri-

bución del progenitor con mayor *fitness*. Además, se incorporó una estrategia selectiva que conserva únicamente el mejor de los dos hijos generados en cada cruce. Ambos cambios, explicados en detalle en la Sección 5.4, buscan aumentar la presión evolutiva y acelerar la convergencia hacia soluciones de mayor calidad, evitando así que soluciones mediocres se propaguen innecesariamente en la población.

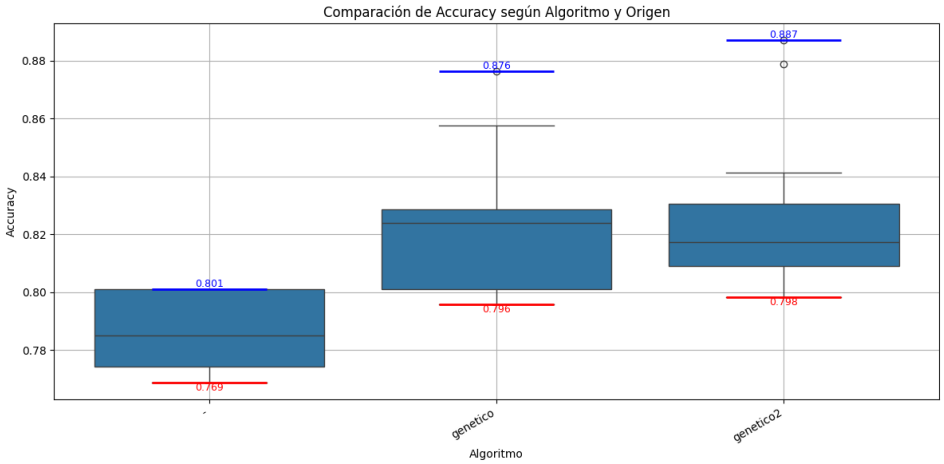

Figura 8.4: Boxplot de *accuracy* para el genético, versión con cruce ponderado y dataset completo.

Tal como se observa en la Figura 8.4, esta modificación produjo una mejora clara en la calidad y estabilidad de los resultados. La versión con cruce ponderado (etiquetada como `genetico2`) alcanza un valor máximo atípico superior (**0.887**), aunque presenta una mediana más baja que la versión básica.

Pero por otra parte, se aprecia una ligera reducción en la dispersión de los valores inferiores, con un mínimo de **0.798** frente al **0.796** en la versión anterior, lo que sugiere una mayor consistencia. Aunque el IQR (Rango Intercuartílico) sigue siendo amplio, la acumulación de valores más cercanos al rango superior refleja una convergencia evolutiva más enfocada y menos dependiente del azar.

En conjunto, esta mejora en el cruce no solo permitió una transferencia más eficiente de características ventajosas, sino que también incrementó la presión selectiva sobre la calidad de las soluciones. Esto se tradujo en un comportamiento más robusto, menos propenso a resultados erráticos y con una mayor capacidad de exploración dirigida del espacio de soluciones.

8.7. Modificación del operador de mutación

A partir de los resultados obtenidos con el algoritmo con cruce ponderado, se evaluó una nueva versión en la que se introdujo una mutación adaptativa (ver Sección 5.5). A diferencia de la versión original con tasa fija, esta estrategia ajusta el número de intercambios en función del tamaño del subconjunto mutado, lo que permite una mayor flexibilidad en escenarios de diferente escala.

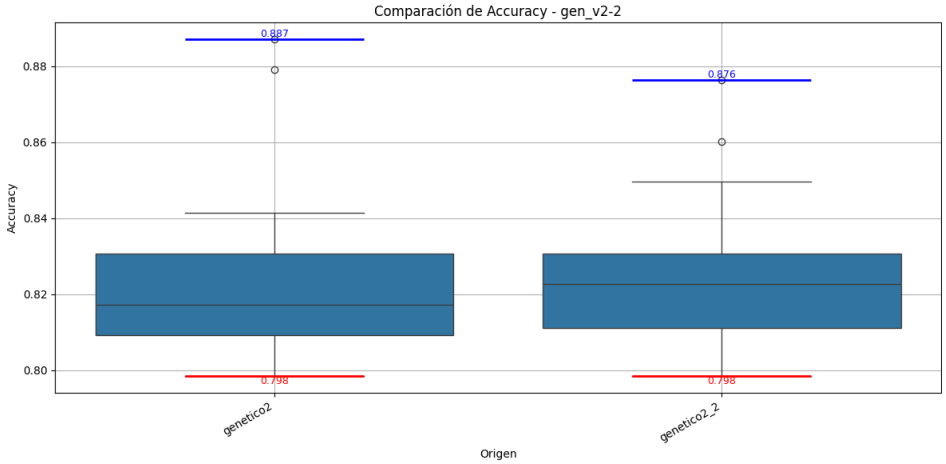

Figura 8.5: Comparación de *accuracy* entre el algoritmo con cruce ponderado (*genetico2*) y su versión con mutación adaptativa (*genetico2_2*).

Como se observa en la Figura 8.5, ambos algoritmos alcanzan un rendimiento similar en términos de mediana y valores extremos, con una ligera ventaja de la versión original (`genetico2`) en el valor máximo de *accuracy* (**0.887** frente a **0.876**). Sin embargo, la versión con mutación adaptativa (`genetico2_2`) muestra una distribución más compacta en la parte central, con menos dispersión hacia valores bajos del IQR, lo que sugiere una mejora en la consistencia entre ejecuciones.

Además, la versión con mutación adaptativa presenta un Q4 más elevado, es decir, su *bigote superior* abarca valores superiores a los de la versión original. Esto indica que, aparte de que la mediana sea un poco superior, un mayor número de ejecuciones alcanzan rendimientos más altos de forma consistente, reforzando la idea de que esta variante promueve una evolución más estable y con mayor concentración de soluciones de calidad en el tramo alto del rendimiento.

Este resultado refleja que, aunque la mutación adaptativa no produce una mejora radical en precisión media, sí contribuye a una evolución más controlada y robusta, reduciendo la probabilidad de degradaciones abruptas.

Además, su comportamiento flexible la hace más adecuada en contextos donde el tamaño del subconjunto varía, evitando configuraciones subóptimas impuestas por una tasa fija.

Aunque el valor máximo de *accuracy* se mantiene en niveles similares, la ganancia está en la consistencia y la eficiencia exploratoria, ya que se adapta automáticamente al contexto sin requerir ajustes manuales para diferentes tamaños de subconjuntos. En conjunto, esta mejora refuerza la capacidad del algoritmo para mantener el equilibrio entre exploración y explotación durante el proceso evolutivo.

8.8. Incorporación del reinicio poblacional

La Versión3 del algoritmo genético (ver Sección 5.6) introduce una lógica de reinicio poblacional diseñada para evitar estancamientos evolutivos. El algoritmo monitoriza el rendimiento del segundo mejor individuo, y si este no mejora durante dos generaciones consecutivas, se aplica un reinicio parcial que conserva únicamente al mejor individuo de la población.

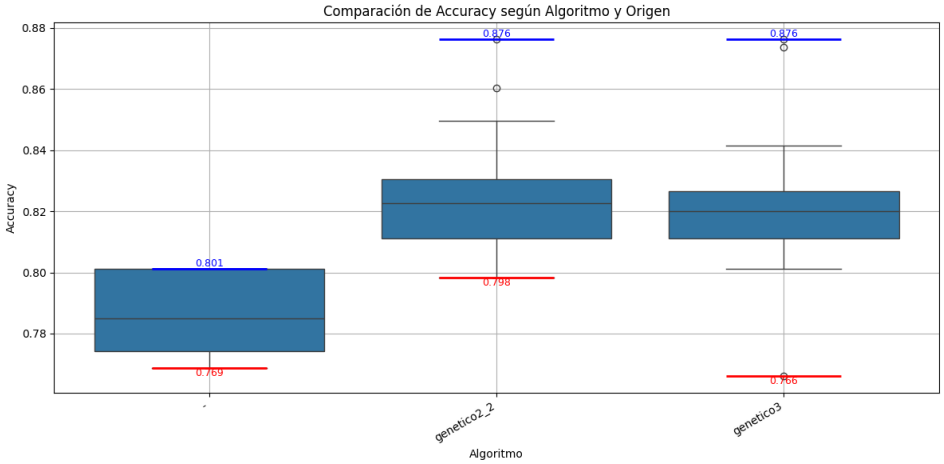

Figura 8.6: Comparación de *accuracy* entre el algoritmo genético con mutación adaptativa (genetico2_2) y con reinicio poblacional (genetico3).

Tal como se observa en la Figura 8.6, los resultados muestran que la incorporación del reinicio no se tradujo en una mejora tangible del rendimiento. La mediana de `genetico3` permanece prácticamente inalterada respecto a `genetico2_2` (incluso ligeramente inferior), y aunque el valor mínimo es inferior (**0.766** frente a **0.798**), no se aprecia un incremento en el valor máximo ni una reducción clara en la dispersión.

Esto sugiere que, al menos en este contexto experimental, el mecanis-

mo de reinicio no logra mejorar la exploración del espacio de soluciones ni mitigar el estancamiento de forma eficaz. En algunos casos, incluso puede introducir una pérdida prematura de diversidad, al regenerar de forma aleatoria parte de la población sin garantizar mejoras sustanciales.

En resumen, aunque el reinicio poblacional es una estrategia teóricamente útil para escapar de óptimos locales, su implementación en esta versión no logró aportar beneficios consistentes en términos de rendimiento.

8.9. Incorporación de versiones libres

Como parte de la evolución de los algoritmos desarrollados, se propusieron versiones libres, en las que el tamaño del subconjunto seleccionado no permanece fijo durante la ejecución, sino que puede ajustarse de forma dinámica.

Para evaluar esta flexibilidad, se generaron versiones libres de la búsqueda local y del algoritmo genético con cruce ponderado y mutación adaptativa, como modificación adicional, se introdujo un mecanismo al algoritmo de búsqueda local en el que se seleccionaba de forma aletarioa el porcentaje inicial de imágenes a partir del cual se comenzaba a trabajar.

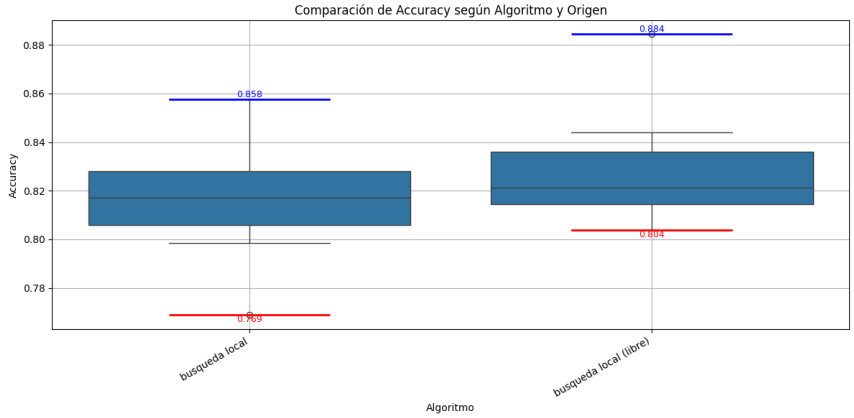

Figura 8.7: Comparación de *accuracy* entre la búsqueda local estándar y su versión libre.

En la Figura 8.7 se observa que la versión libre de la búsqueda local mejora tanto la mediana como el valor máximo de *accuracy* respecto a su versión original. Presenta, además, una menor dispersión en los valores bajos y una mayor concentración de ejecuciones en torno al cuartil superior, lo que sugiere una mejora en la estabilidad del algoritmo. Este comportamiento refleja que introducir flexibilidad en el tamaño del subconjunto aporta capacidad

de adaptación sin comprometer la consistencia del rendimiento.

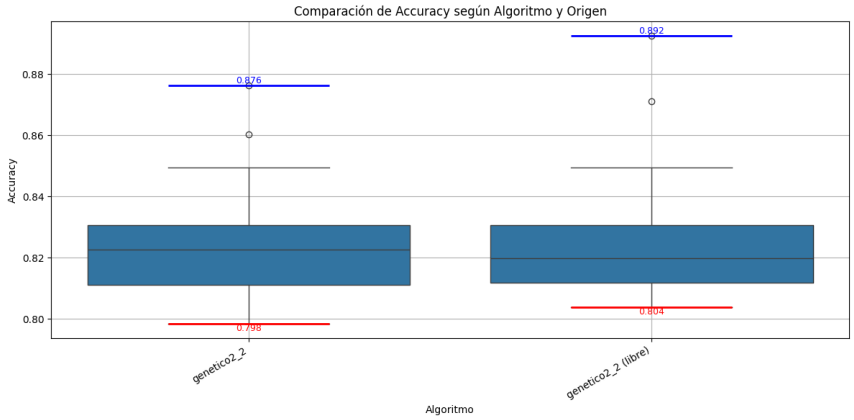

Figura 8.8: Comparación de *accuracy* entre el algoritmo genético v2-2 (cruce ponderado + mutación adaptativa) y su versión libre.

En el caso del genético con cruce ponderado y mutación adaptativa (Figura 8.8), los resultados son algo más matizados: la versión libre mantiene un rendimiento muy similar al de la versión rígida, con una leve mejora en los valores altos y un mínimo más elevado. Esto sugiere que, aunque no siempre se observe una mejora significativa en mediana, la capacidad adaptativa aporta robustez adicional y mayor protección frente a ejecuciones fallidas.

Falta añadir boxplot que compare el accuracy separado por cada porcentaije inicail del algo

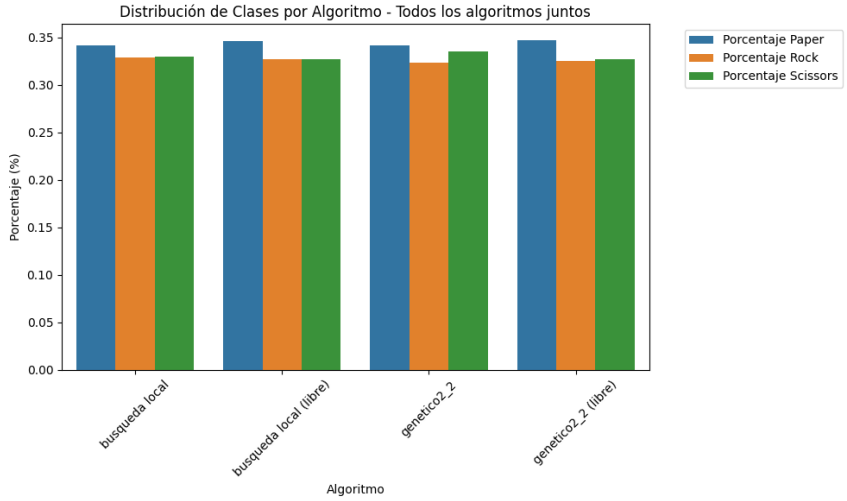

Figura 8.9: Distribución de clases en los subconjuntos generados por los algoritmos estándar y libres.

La Figura 8.9 muestra que tanto los algoritmos estándar como sus variantes libres mantienen una distribución equilibrada entre las tres clases del conjunto `RPS`. Las proporciones de `Rock`, `Paper` y `Scissors` se reproducen de forma muy similar en todos los casos, con únicamente pequeñas variaciones que no resultan significativas como para suponer un desbalance.

Este resultado es especialmente relevante para la validez de los experimentos, ya que confirma que los mecanismos evolutivos y las estrategias de ajuste del tamaño del subconjunto no introducen sesgos de clase. La representatividad estructural del conjunto se conserva, lo cual es fundamental para garantizar una evaluación justa y equilibrada en tareas de clasificación multiclase.

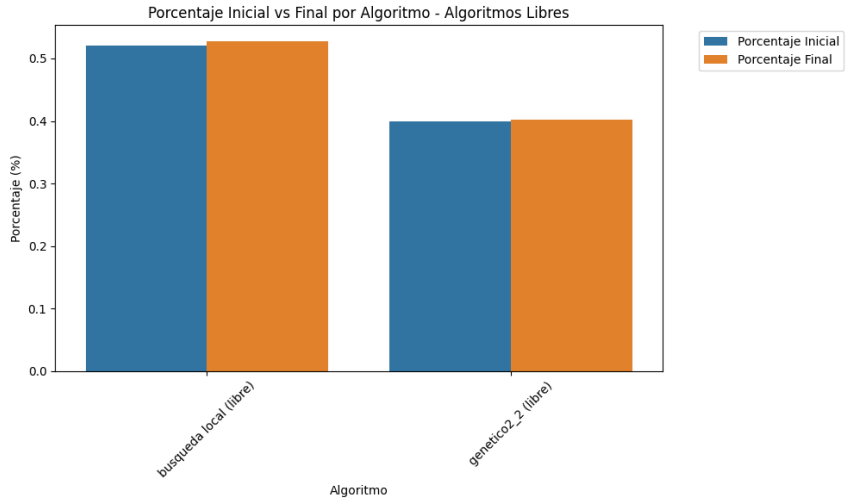

Figura 8.10: Comparación entre porcentaje inicial y final en los algoritmos libres.

Las Figuras 8.10 y 8.11 permiten analizar con mayor detalle cómo se comportan los algoritmos libres en términos de tamaño del subconjunto seleccionado. En la comparación directa por algoritmo (Figura 8.10), se observa que el porcentaje final tiene un leve incremento respecto al inicial.

Sin embargo, al desglosar el análisis por valores específicos del porcentaje inicial (Figura 8.11), se aprecia un patrón más claro: cuanto mayor es el punto de partida, mayor tiende a ser también el porcentaje final alcanzado. Esta tendencia es especialmente marcada en el caso de la búsqueda local libre, que muestra una progresión prácticamente lineal. El algoritmo genético, en cambio, presenta un crecimiento más escalonado, pero igualmente adaptativo.

Este comportamiento sugiere que los algoritmos libres no solo adaptan la composición del subconjunto, sino también su escala, ajustando dinámica-

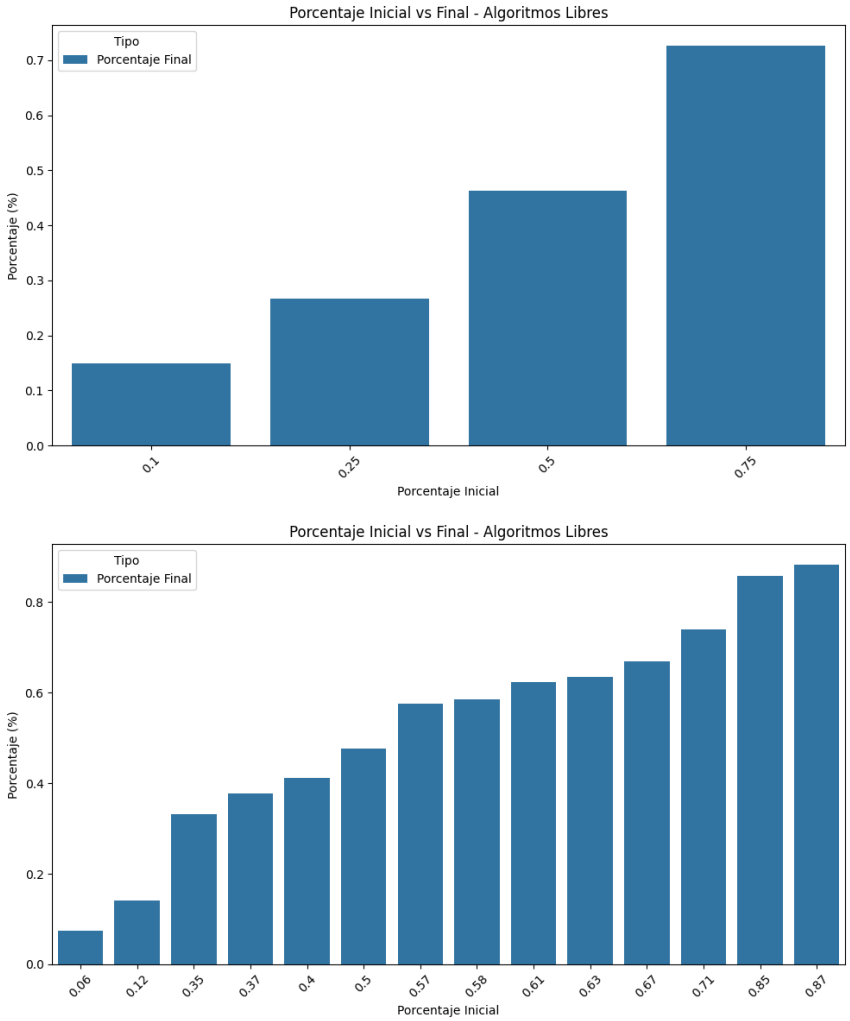

Figura 8.11: Porcentaje final alcanzado en función del porcentaje inicial. Arriba: genético libre. Abajo: búsqueda local libre.

mente la cantidad de datos seleccionados según el entorno de búsqueda. En general, estas versiones demuestran ser una opción más versátil y controlada, especialmente útil cuando no se conoce de antemano el tamaño óptimo del conjunto de entrenamiento.

8.10. Evaluación del algoritmo memético

Finalmente, se evaluó el algoritmo memético (ver Sección 5.7), el cual combina la evolución genética con una búsqueda local aplicada de forma probabilística sobre ciertos individuos seleccionados. Este enfoque híbrido busca equilibrar la exploración del espacio de soluciones con una intensificación localizada, ofreciendo mejoras tanto en precisión como en estabilidad.

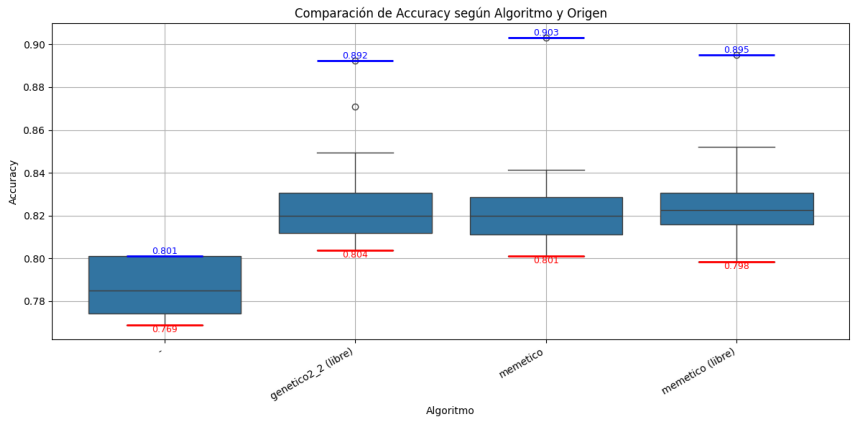

Figura 8.12: Comparación de *accuracy* entre el algoritmo memético estándar y su versión libre.

Tal como se aprecia en la Figura 8.12, el algoritmo memético supera claramente al mejor de los enfoques genéticos (el Genetico Libre con Cruce Ponderado y Mutación Adaptativa), alcanzando una mediana más elevada y un valor máximo de *accuracy* de hasta **0.903**. Su distribución es más compacta, con menor dispersión hacia los valores bajos, lo que refleja una mayor consistencia entre ejecuciones.

La versión libre del memético, que incorpora también un ajuste dinámico del tamaño del subconjunto (ver Apartado 5.8.3), muestra un rendimiento muy similar al estándar, con una ligera reducción en el valor máximo pero una estabilidad comparable. Esto sugiere que el componente adaptativo no penaliza la calidad de las soluciones y puede incluso aportar mayor flexibilidad en entornos más inciertos.

En la Figura 8.13 se observa cómo el memético libre tiende a incrementar

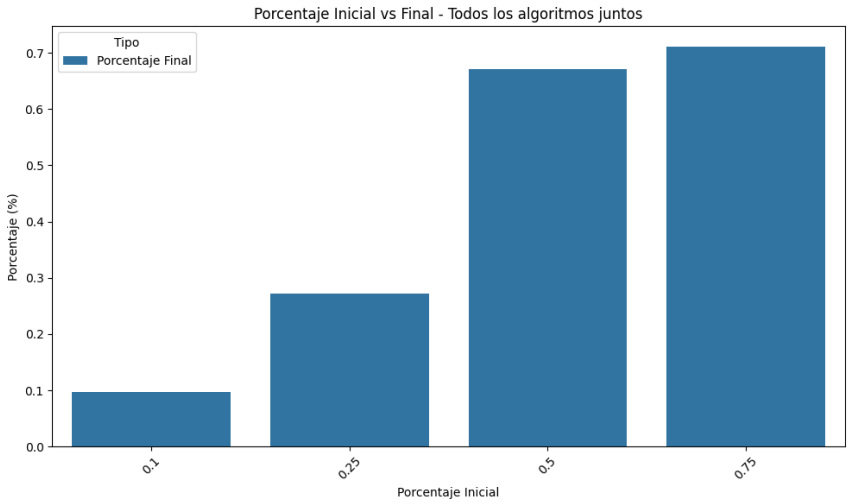

Figura 8.13: Porcentaje final alcanzado por el algoritmo memético libre en función del porcentaje inicial.

el tamaño del subconjunto conforme aumenta el porcentaje inicial. La curva es progresiva y cercana a la linealidad, lo que indica un comportamiento estructuralmente coherente: el algoritmo detecta cuándo puede beneficiarse de incluir más ejemplos y adapta su escala en consecuencia. Este ajuste dinámico refuerza su capacidad para equilibrar rendimiento y tamaño, maximizando la eficiencia de la selección sin depender de una configuración fija.

De esta forma, tanto el memético estándar como su variante libre se consolidan como las estrategias más eficaces del estudio.

8.11. Comparación final entre enfoques

En esta sección se realiza una comparación conjunta entre todos los algoritmos implementados. Se presentan boxplots y gráficos de barras que muestran la evolución del *accuracy*, el porcentaje de datos utilizados y el equilibrio entre clases.

Falta realizar la comparación final.

8.12. Validación con el dataset `PAINTING`

Para comprobar la robustez de los algoritmos desarrollados, se validaron los experimentos con el dataset `PAINTING`, caracterizado por una mayor

complejidad visual y un número superior de clases respecto al conjunto `RPS`. Con el fin de mantener un equilibrio entre complejidad y coste computacional, se limitaron los experimentos a los porcentajes iniciales del 25 % y 50 %, evaluando únicamente los algoritmos más representativos.

En particular, se seleccionó el algoritmo memético, junto con su versión libre, ya que ambos habían demostrado ser los más eficaces en las pruebas anteriores. Para establecer una referencia clara, se incluyó también los resultados obtenidos usando el 100 % del conjunto de datos, y para la compración de *accuracy* entre algoritmos también se añadió el resultado del algoritmo aleatorio.

8.12.1. Comparación de *accuracy* entre algoritmos

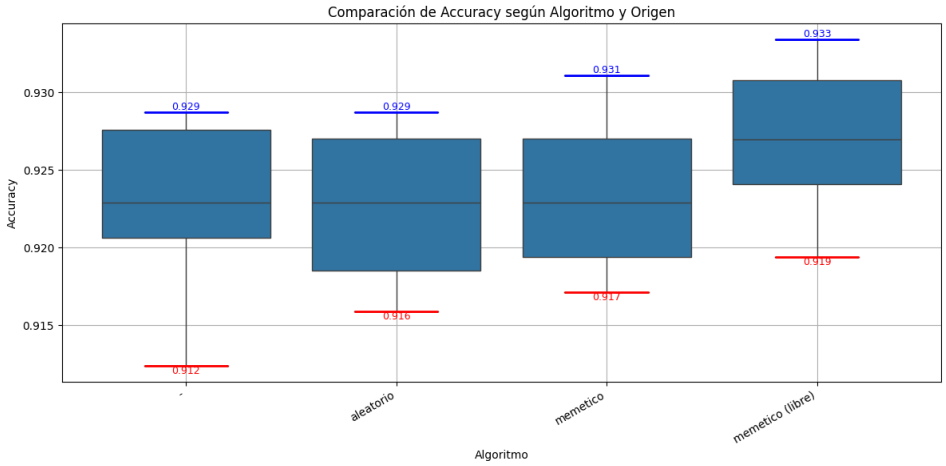

Figura 8.14: Boxplot comparando resultados con el dataset `PAINTING` usando *accuracy*.

En la Figura 8.14 se comparan los valores de *accuracy* obtenidos al aplicar distintos algoritmos de selección de subconjuntos en el conjunto de datos `PAINTING`. A simple vista, destaca el buen rendimiento de los tres enfoques meméticos frente al uso directo del 100 % del conjunto o la selección aleatoria, lo cual es especialmente significativo al tratarse de un conjunto con alta complejidad visual y estructural.

El algoritmo memético libre obtiene el mejor rendimiento general, con una mediana que ronda el **0.927** y un valor máximo de **0.933**, superando incluso la ejecución con el 100 % de los datos, cuyo máximo se sitúa en **0.929**. Esto no solo refuerza su capacidad de generalización, sino que demuestra que una selección optimizada puede superar a la totalidad del conjunto original, posiblemente por eliminar ejemplos redundantes o incluso perjudiciales.

El algoritmo memético estándar también muestra una precisión elevada, con una mediana apenas inferior al libre, y un máximo de **0.931**. Sin embargo, su varianza es ligeramente mayor, lo que sugiere que la falta de ajuste dinámico del tamaño del subconjunto puede limitar su adaptabilidad en ciertas ejecuciones.

Por otro lado, la selección aleatoria, aunque muestra valores aceptables, vuelve a confirmar su principal debilidad: la alta dispersión. Con un mínimo de **0.916** y una mediana prácticamente idéntica a la obtenida con el uso completo del dataset, su comportamiento se sitúa como una referencia básica pero poco fiable. Puede alcanzar buenos resultados, pero lo hace de forma inconsistente y sin mecanismos que garanticen estabilidad.

La ejecución con el **100 %** de los datos, utilizada como referencia absoluta, queda por debajo de las soluciones obtenidas por los algoritmos meméticos. Esto valida empíricamente que no es la cantidad de datos, sino su calidad y representatividad, lo que determina la eficacia del entrenamiento en entornos complejos.

En resumen, los algoritmos meméticos, especialmente en su versión libre, muestran el mejor rendimiento en términos de precisión y estabilidad, superando al uso del 100 % de los datos y al enfoque aleatorio.

8.12.2. Impacto del porcentaje inicial

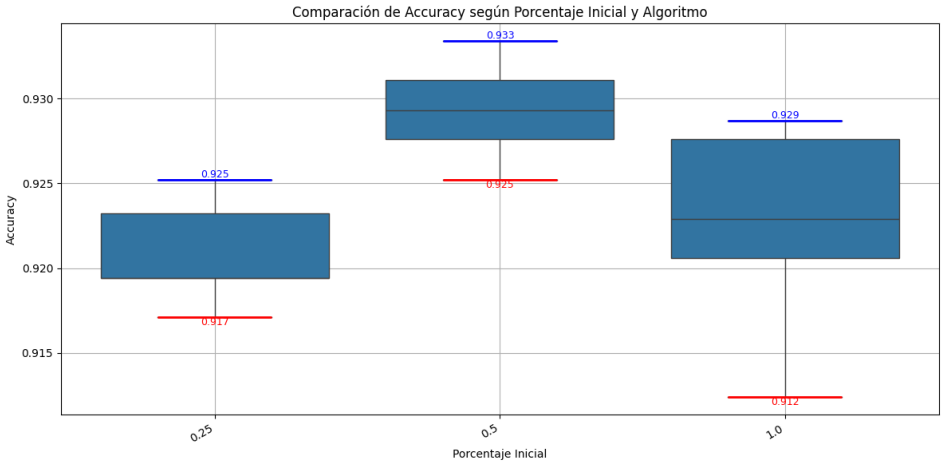

Figura 8.15: Boxplot de *accuracy* según el porcentaje inicial de datos.

La Figura 8.15 muestra cómo varía el rendimiento del modelo (medido en términos de *accuracy*) al utilizar diferentes porcentajes iniciales de datos para los algoritmos meméticos en el dataset PAINTING. Lo primero que destaca es que el uso del 50 % del conjunto original no solo logra un rendi-

miento equiparable, sino incluso superior al uso del 100 %. Con una mediana cercana a **0.930** y un máximo de **0.933**, este porcentaje logra un equilibrio ideal entre compresión de datos y preservación de información relevante.

Este resultado sugiere que, a partir de cierto umbral, añadir más datos no solo deja de aportar valor, sino que puede introducir ruido o redundancia. En este caso, el entrenamiento con el 100 % de los datos exhibe mayor dispersión y un mínimo significativamente más bajo (**0.912**), lo cual indica una mayor variabilidad entre ejecuciones y una menor estabilidad general.

Por otro lado, el uso del 25 % inicial también demuestra un rendimiento sorprendentemente competitivo, alcanzando un máximo de **0.925**. Sin embargo, su rango intercuartílico más estrecho (menor dispersión de los resultados) y su menor valor mínimo (**0.917**) evidencian una cierta fragilidad ante la reducción excesiva: puede funcionar bien si la selección es óptima, pero el margen de error es menor.

Este comportamiento ilustra un fenómeno interesante en la reducción de datos: no existe una relación lineal entre cantidad de datos y precisión, sino que el valor está en la calidad y representatividad del subconjunto. El resultado más robusto refuerza la hipótesis de que existe un punto de saturación a partir del cual la adición de ejemplos tiene un efecto marginal o incluso contraproducente en modelos convolucionales.

En resumen, los resultados respaldan la viabilidad de utilizar una fracción cuidadosamente seleccionada de datos para obtener resultados comparables o incluso superiores al uso del conjunto completo.

8.12.3. Equilibrio en la distribución de clases

La Figura 8.16 muestra la proporción de clases preservada por cada algoritmo durante el proceso de reducción. Se observa que tanto el algoritmo memético estándar como el libre mantienen una distribución prácticamente idéntica a la original, sin introducir sesgos estructurales.

Las clases mayoritarias (`Iconography`, `Painting` y `Sculpture`) se mantienen consistentes en su proporción, al igual que las clases minoritarias (`Drawings` y `Engraving`). Esta conservación sugiere que el proceso de selección no actúa de forma aleatoria ni ciega, sino que incorpora implícitamente una presión hacia la diversidad, lo cual es clave en problemas multiclase.

8.12.4. Evolución del tamaño del subconjunto

En la Figura 8.17 se analiza cómo evolucionó el tamaño de los subconjuntos seleccionados por los algoritmos libres respecto al valor inicial. Se

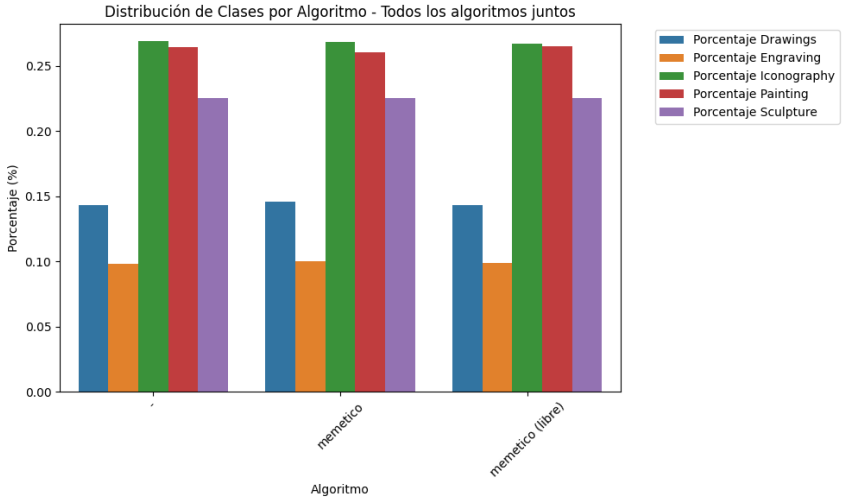

Figura 8.16: Distribución de clases seleccionadas por cada algoritmo.

observa que el algoritmo memético libre tiende a incrementar el porcentaje de imágenes seleccionadas durante el proceso evolutivo.

Este comportamiento adaptativo pone de manifiesto la capacidad del algoritmo para ajustar dinámicamente la escala de la solución en función de la complejidad del problema. A diferencia de enfoques con tamaño fijo, el algoritmo libre no solo decide qué ejemplos seleccionar, sino también cuántos, ampliando el conjunto cuando detecta que puede mejorar el rendimiento sin incurrir en sobreajuste.

Además, se observa que al partir del 25 % inicial, el algoritmo tiende a aumentar el tamaño del subconjunto hasta un 16,35 % adicional, en cambio, al partir del 50 % inicial, el crecimiento es mayor, alcanzando un 18,69 % adicional [1]. Este hecho puede sugerir una cierta prudencia evolutiva: el algoritmo no se expande indiscriminadamente, sino que responde a señales de mejora en el proceso de evaluación. Este mecanismo emergente refuerza la versatilidad del enfoque libre, que no solo busca soluciones de alta calidad, sino que lo hace optimizando también el volumen de datos utilizados.

Síntesis final de la validación con PAINTING

Los resultados obtenidos con el dataset PAINTING confirman la solidez y capacidad de generalización de los algoritmos desarrollados. Tanto el enfoque memético estándar como, especialmente, su versión libre, lograron mante-

[1] Porcentajes exactos sacados de la tabla de resultados.

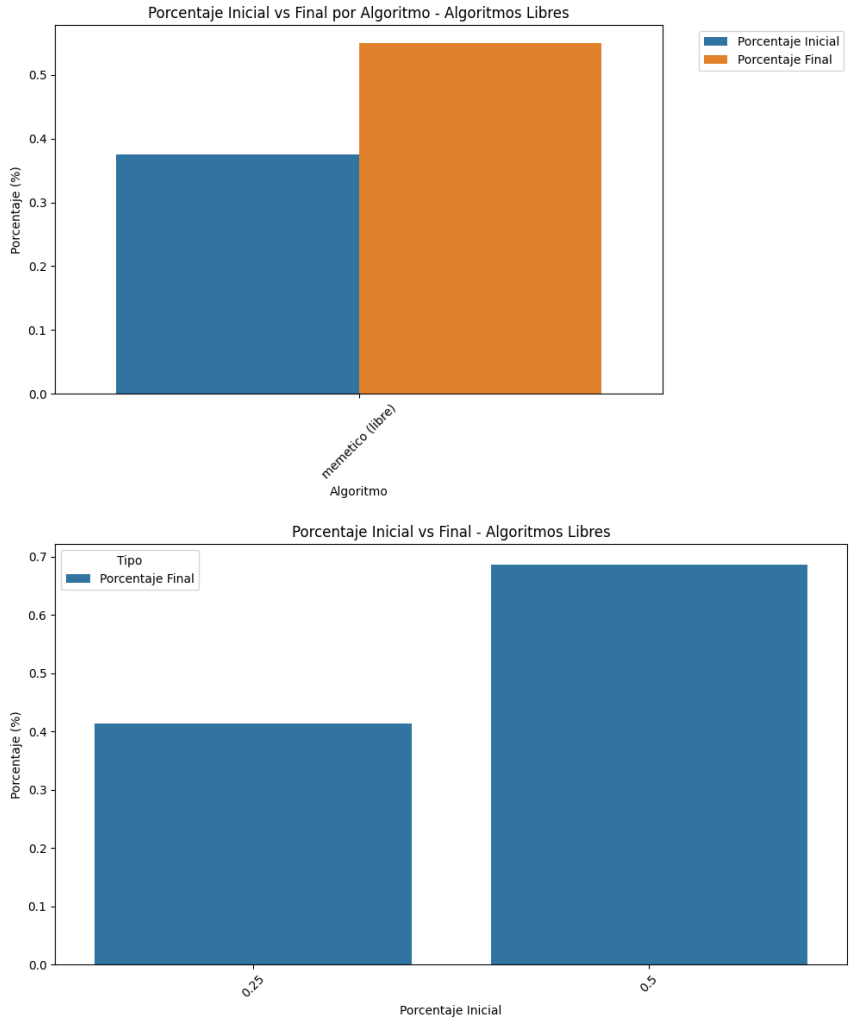

Figura 8.17: Evolución del tamaño del subconjunto seleccionado por los algoritmos libres. Arriba: comparación por algoritmo. Abajo: evolución según el porcentaje inicial.

ner un rendimiento competitivo en un entorno más complejo y con mayor número de clases.

Se evidenció que es posible superar el rendimiento del conjunto completo de entrenamiento utilizando únicamente un subconjunto bien seleccionado, reduciendo significativamente el volumen de datos sin comprometer la precisión. Además, los algoritmos meméticos demostraron preservar el equilibrio entre clases y adaptarse dinámicamente al tamaño del subconjunto, lo que constituye una ventaja clave en tareas reales donde no siempre se dispone de datasets perfectamente balanceados ni completos.

En conjunto, esta validación externa no solo refuerza las conclusiones obtenidas con RPS, sino que demuestra que el uso de técnicas evolutivas, y en particular las variantes libres, puede ser una alternativa eficaz, escalable y controlable frente a la selección masiva o aleatoria de datos en procesos de entrenamiento profundo.

Capítulo 9

Conclusiones

9.0.1. Reflexión sobre el ajuste progresivo

El desarrollo iterativo de estas versiones no solo mejoró los resultados, sino que permitió experimentar con distintos enfoques de convergencia, diversidad poblacional y equilibrio entre exploración y explotación.

En definitiva, el algoritmo memético y las versiones v2 y v3 del algoritmo genético se consolidaron como los más robustos para este problema, logrando reducciones significativas del conjunto de entrenamiento sin sacrificar precisión, y con tiempos de entrenamiento razonables gracias a su capacidad de selección eficiente.

Capítulo 10

Bibliografía

[1] *Scrum Guide*, https://scrumguides.org/scrum-guide.html. visitado 25 de feb. de 2025.

[2] *Notion - Gestión de Tareas*, https://www.notion.com/es-es/help/guides/personal-work-dashboard. visitado 27 de feb. de 2025.

[3] *Enhancing collaboration in project-based organizations with it*, https://www.pmi.org/learning/library/enhancing-collaboration-project-based-organizations-7141. visitado 11 de mayo de 2025.

[4] S. Chacon y B. Straub, *Pro Git*, Second. New York: Apress, 2014, https://git-scm.com/book/en/v2, ISBN: 978-1-4842-0076-6. DOI: 10.1007/978-1-4842-0076-6.

[5] *Salario para Data Scientist en España - Salario Medio*, https://es.talent.com/salary. visitado 24 de feb. de 2025.

[6] *Overview Google Cloud*, https://cloud.google.com/docs/overview?hl=es-419. visitado 25 de feb. de 2025.

[7] *What is cloud run | cloud run documentation*, https://cloud.google.com/run/docs/overview/what-is-cloud-run. visitado 25 de feb. de 2025.

[8] S. Weidman, *Deep Learning from Scratch: Building with Python from First Principles*. O'Reilly Media, Incorporated, 2019, https://books.google.es/books?id=PRSCwwEACAAJ, ISBN: 978-1-4920-4141-2.

[9] P. H. Sydenham y R. Thorn, eds., *Handbook of measuring system design*. Chichester: Wiley, 2005, ISBN: 978-0-470-02143-9.

[10] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci y M. Parmar, "A review of convolutional neural networks in computer vision," *Artificial Intelligence Review*, vol. 57, n.º 4, pág. 99, mar. de 2024, https://doi.org/10.1007/s10462-024-10721-6, ISSN: 1573-7462. DOI: 10.1007/s10462-024-10721-6.

[11] *Resnet50*, https://pytorch.org/hub/nvidia_deeplearningexamples_resnet50/. visitado 24 de feb. de 2025.

[12] K. He, X. Zhang, S. Ren y J. Sun, "Deep Residual Learning for Image Recognition," en *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, págs. 770-778. DOI: 10.1109/CVPR.2016.90.

[13] A. Alnuaim, M. Zakariah, W. A. Hatamleh, H. Tarazi, V. Tripathi y E. T. Amoatey, "Human-computer interaction with hand gesture recognition using resnet and mobilenet," *Computational Intelligence and Neuroscience*, vol. 2022, n.º 1, pág. 8 777 355, 2022, https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/8777355, ISSN: 1687-5273. DOI: 10.1155/2022/8777355.

[14] A. G. Howard et al., *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, http://arxiv.org/abs/1704.04861, abr. de 2017. DOI: 10.48550/arXiv.1704.04861. arXiv: 1704.04861 [cs].

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov y L.-C. Chen, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, http://arxiv.org/abs/1801.04381, mar. de 2019. DOI: 10.48550/arXiv.1801.04381. arXiv: 1801.04381 [cs].

[16] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016.

[17] Y. LeCun, Y. Bengio y G. Hinton, "Deep learning," *Nature*, vol. 521, n.º 7553, págs. 436-444, mayo de 2015, https://doi.org/10.1038/nature14539, ISSN: 1476-4687. DOI: 10.1038/nature14539.

[18] C. Shorten y T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, n.º 1, pág. 60, jul. de 2019, https://doi.org/10.1186/s40537-019-0197-0, ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0.

[19] J. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975, https://books.google.es/books?id=YE5RAAAAMAAJ, ISBN: 978-0-472-08460-9.

[20] P. Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms," *Caltech Concurrent Computation Program*, oct. de 2000.

[21] F. Neri, C. Cotta, P. Moscato y J. Kacprzyk, eds., *Handbook of Memetic Algorithms* (Studies in Computational Intelligence). Berlin, Heidelberg: Springer, 2012, vol. 379, http://link.springer.com/10.1007/978-3-642-23247-3, ISBN: 978-3-642-23246-6 978-3-642-23247-3. DOI: 10.1007/978-3-642-23247-3.

[22] J. Dong, L. Zhang, B. Hou y L. Feng, "A Memetic Algorithm for Evolving Deep Convolutional Neural Network in Image Classification," dic. de 2020, págs. 2663-2669. DOI: 10.1109/SSCI47803.2020.9308162.

[23] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison Wesley series in artificial intelligence). Addison-Wesley, 1989, https://books.google.es/books?id=2IIJAAAACAAJ, ISBN: 978-0-201-15767-3.

[24] J. VanderPlas, *Python Data Science Handbook: Essential Tools for Working with Data*. ."O'Reilly Media, Inc.", nov. de 2016, https://books.google.es/books?hl=es&lr=&id=xYmNDQAAQBAJ&oi=fnd&pg=PR2&dq=Python+Data+Science+Handbook&ots=Xs9Rj3qj-M&sig=f5K5ixzKjH7pc2Uo2IYW9jrPNI8#v=onepage&q=Python%20Data%20Science%20Handbook&f=false, ISBN: 978-1-4919-1214-0.

[25] N. Ketkar y J. Moolayil, "Introduction to PyTorch," en *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, https://doi.org/10.1007/978-1-4842-5364-9_2, Berkeley, CA: Apress, 2021, págs. 27-91, ISBN: 978-1-4842-5364-9. DOI: 10.1007/978-1-4842-5364-9_2.

[26] *torch.cuda — PyTorch 2.4 documentation*, https://pytorch.org/docs/stable/cuda.html. visitado 8 de oct. de 2024.

[27] O. Kramer, "Scikit-Learn," en *Machine Learning for Evolution Strategies*, https://doi.org/10.1007/978-3-319-33383-0_5, Cham: Springer International Publishing, 2016, págs. 45-53, ISBN: 978-3-319-33383-0. DOI: 10.1007/978-3-319-33383-0_5.

[28] *NumPy v2.0 Manual*, https://numpy.org/doc/2.0/index.html. visitado 24 de feb. de 2025.

[29] *Polars — Python API reference*, https://docs.pola.rs/api/python/stable/reference/. visitado 24 de feb. de 2025.

[30] *Matplotlib 3.9.3 documentation*, https://matplotlib.org/3.9.3/index.html. visitado 24 de feb. de 2025.

[31] *Seaborn 0.13.2 documentation*, https://seaborn.pydata.org/tutorial.html. visitado 3 de mayo de 2025.

[32] *Openpyxl 3.1.3 documentation*, https://openpyxl.readthedocs.io/en/stable/. visitado 3 de mayo de 2025.

[33] *Creation of virtual environments*, https://docs.python.org/3/library/venv.html. visitado 24 de feb. de 2025.

[34] *Conda Documentation*, https://docs.conda.io/en/latest/. visitado 24 de feb. de 2025.

[35] *cuBLAS Deterministic Algorithms*, https://docs.nvidia.com/cuda/cublas/index.html#cublasApi_reproducibility. visitado 24 de feb. de 2025.

[36] *Early Stopping Discussion*, https://www.geeksforgeeks.org/early-stopping-on-validation-loss-or-on-accuracy/, feb. de 2024. visitado 4 de mayo de 2025.

[37] *Rock Paper Scissors Dataset*, https://public.roboflow.com/classification/rock-paper-scissors. visitado 24 de feb. de 2025.

[38] *(original) art images: Drawing/painting/sculptures/engravings*, https://www.kaggle.com/datasets/thedownhill/art-images-drawings-painting-sculpture-engraving. visitado 24 de feb. de 2025.

[39] *(cleaned) art images: Drawing/painting/sculptures/engravings*, https://www.kaggle.com/datasets/moosecat/art-images-drawings-painting-sculpture-engraving. visitado 24 de feb. de 2025.

Capítulo 11

Bibliografía de Referencia[1]

[1]Este capítulo contiene las referencias bibliográficas consultadas (que no han sido citadas en el cuerpo del documento).