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, Noviembre de 2024

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... y, además, 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 un número extremadamente alto 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, la reciente **legislación europea sobre IA** (IA Act) [1] establece 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**... sin comprometer la calidad del modelo.

Ya se ha demostrado que aumentar el número de imágenes de entrenamiento puede, en ciertos casos, mejorar el proceso; sin embargo, esto solo sucede cuando las imágenes adicionales realmente contribuyen al aprendizaje. De hecho, las **técnicas de aumento de datos** (Data Augmentation) permiten reducir la necesidad de muchas imágenes similares, ya que estas pueden generarse de manera automática a partir de las ya existentes... lo que además evita problemas legales asociados a la autoría de los datos.

En este trabajo, proponemos el uso de **algoritmos meméticos** combinados con **métricas de similitud entre imágenes** 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 con una calidad 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 **Artificial Intelligence**, as they are capable of processing large volumes of data... and, moreover, recognizing highly complex patterns. Among these, **convolutional models** have proven to be particularly effective in identifying objects and characteristics in images — an essential capability for many modern applications. However, unlike humans, these models require an extremely high number of training data for each category they need to learn. This implies a longer training process, and often, the collection of the necessary data can be problematic, depending on the type of information required.

In addition to the difficulty of obtaining data, the recent **European AI legislation** (IA Act) [1] establishes the need to audit not only the models but also the data used to train them, especially when dealing with AI applications that handle sensitive data. These audits, by their very nature, will become more complex as the size of the training set increases. Therefore, it becomes completely necessary to develop strategies that allow for **reducing the size of training datasets**... without compromising the quality of the model.

It has already been demonstrated that increasing the number of training images can, in certain cases, improve the process; however, this only happens when the additional images actually contribute to learning. In fact, **Data Augmentation techniques** help reduce the need for many similar images, as these can be automatically generated from existing ones... which also avoids legal issues related to data authorship.

In this work, we propose the use of **memetic algorithms**, combined with **image similarity metrics**, to establish a process of **training dataset reduction** — known as **instance selection**. The idea is to select

a reduced set of representative images that, along with data augmentation techniques, are sufficient to train convolutional models with optimal quality. In this way, the size of the training dataset could be significantly reduced, while maintaining the quality of learning and, at the same time, facilitating both the auditing process and the computational efficiency of the system.

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 X de mes de 201 .

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 X de mes de 201 .

Los directores:

Daniel Molina Cabrera (tutor)

Agradecimientos

Poner aquí agradecimientos...

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 grandes cantidades 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 representa una barrera considerable para muchos entornos, 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 de subconjuntos representativos se presenta como una solución 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 resulta fundamental 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 cumplir con este objetivo general, se plantean los siguientes **objetivos específicos**:

- **Desarrollar** e implementar metaheurísticas que busquen conjuntos reducidos de datos de entrenamiento, con el fin de reducir el volumen de datos requerido para entrenar modelos convolucionales —sin comprometer la precisión de los resultados—.
- **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—.
- **Mejorar la eficiencia** del entrenamiento de redes neuronales convolucionales mediante el uso de metaheurísticas, analizando los beneficios en términos de reducción de tiempo y costo computacional.
- **Contribuir al avance** de soluciones innovadoras en el campo del aprendizaje profundo, especialmente en escenarios con limitaciones de datos; facilitando así el acceso a esta tecnología a sectores que, de otro modo, tendrían dificultades para implementarla de manera efectiva.

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.

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** [2], ampliamente utilizado en entornos donde se requiere adaptación constante y entregas progresivas. Al tratarse de un trabajo de investigación que ha ido evolucionando a medida que se obtenían resultados, este enfoque resultó adecuado para mantener una estructura flexible y avanzar de forma iterativa.

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 gestionar las tareas de forma ordenada, se utilizó **Notion** [3] como herramienta personal de planificación. 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, que se repetían 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 parámetros si era 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

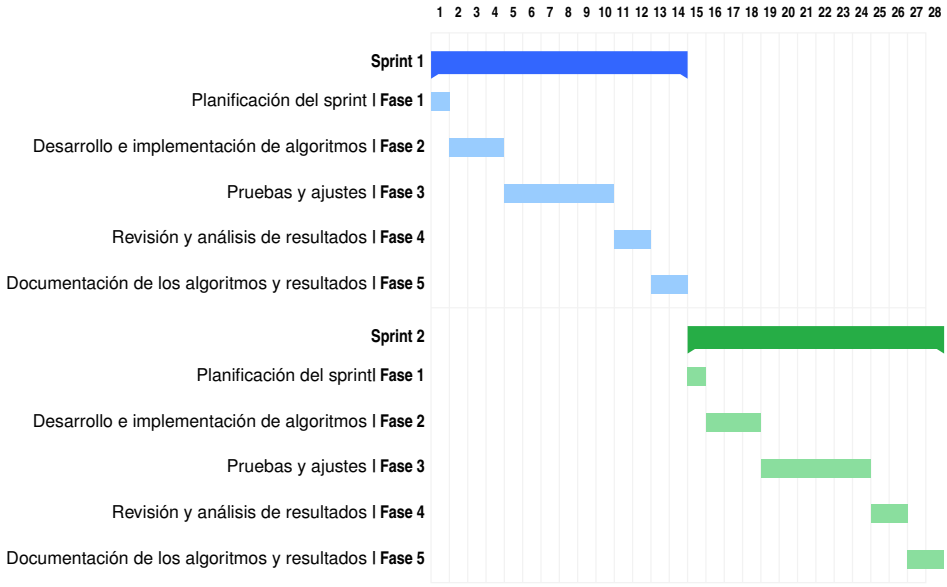

Figura 2.1: Diagrama de Gantt que muestra la planificación de dos sprints del proyecto, con sus respectivas fases: planificación, desarrollo, pruebas, revisión y documentación. Este esquema ilustra la estructura iterativa seguida durante el desarrollo.

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**, fue posible reorganizar prioridades y redistribuir tareas para poder llegar a los objetivos 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 trabajo total estimado es de **400 horas**, repartidas de manera flexible a lo largo de los ciclos. El coste por hora se ha estimado en **20 euros** [5], lo cual incluye las tareas de investigación, desarrollo de algoritmos, análisis de resultados y documentación.

Ciclo de trabajo	**Horas dedicadas**	**Coste por hora (€)**	**Coste total (€)**
Planificación del sprint	30	20	600
Desarrollo e implementación de algoritmos	80	20	1.600
Pruebas y ajustes	180	20	3.600
Revisión y análisis de resultados	55	20	1.100
Documentación de algoritmos y resultados	55	20	1.100
Total	**400 horas**		**8.000 €**

Tabla 2.1: Coste estimado de la mano de obra basado en los ciclos de trabajo.

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. Por ello, para suponer un coste estimado, se ha supuesto lo que nos costaria un servidor de **Google Cloud** [6] de **Compute Engine** [7] en Bélgica.

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 manerá 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: Coste estimado de los recursos computacionales.

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 generado costes asociados a licencias de software.

2.2.4. Otros gastos

Se han considerado gastos adicionales, como el uso de **conexión a internet** y el consumo de **electricidad** durante el desarrollo y entrenamiento de los modelos.

Concepto	Coste estimado (€)
Conexión a internet	50
Electricidad	40
Total	**90 €**

Tabla 2.3: Otros gastos estimados.

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
Otros gastos	90
Total	**8.600 €**

Tabla 2.4: Presupuesto total estimado del proyecto.

Capítulo 3

Fundamentos de Aprendizaje Profundo

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**:

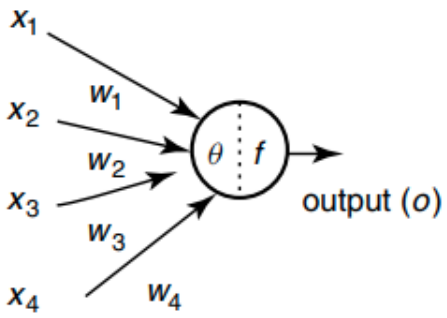

Figura 3.1:

   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:

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

   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**:

   - **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.

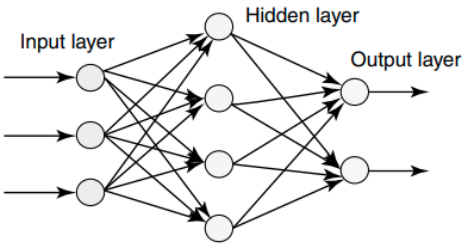

Figura 3.2:

- **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

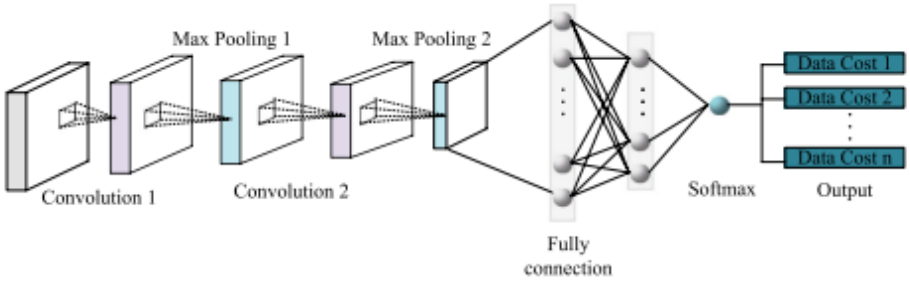

Figura 3.3: En esta imagen extraída de [10] puede observarse de la estructura de una red neuronale convolucional. Junto a sus capas convolucionales.

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.

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

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**

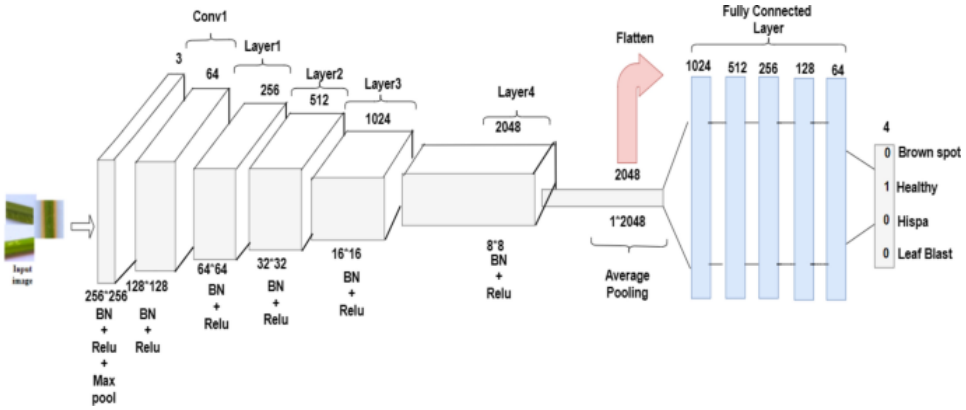

Figura 3.4: Diagrama de la Arquitectuda de ResNet50

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

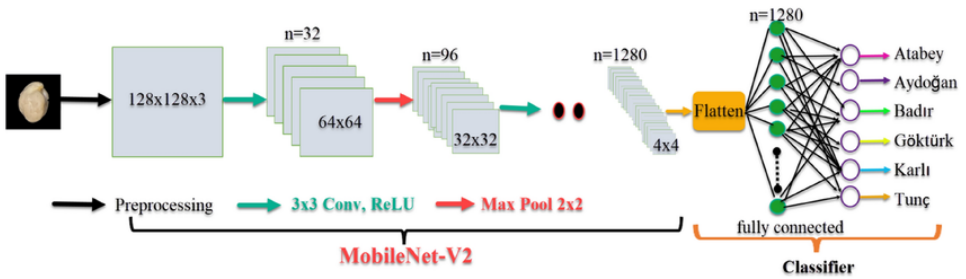

Figura 3.5: Diagrama de la Arquitectuda de MobileNetV2

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 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. 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.

5.1.3. 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.

El proceso básico de la búsqueda local es el siguiente:

1. Se genera una solución inicial, por ejemplo, seleccionando un subconjunto de datos aleatoriamente.

2. Se realizan cambios locales en la solución, como añadir o eliminar ejemplos del conjunto de datos.

3. Se evalúa la nueva solución según el rendimiento del modelo de aprendizaje profundo.

4. Si la nueva solución es mejor, se reemplaza la solución actual por esta.
5. El proceso se repite hasta que no se observan mejoras significativas o hasta que se alcanza un número
6. predefinido de iteraciones.

5.2.2. 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.

5.2.3. 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. Algoritmos Genéticos

Los **algoritmos genéticos** son algoritmos de búsqueda inspirados en los principios de la evolución natural. En este trabajo, se aplican con el objetivo de encontrar subconjuntos óptimos de datos de entrenamiento, reduciendo el tamaño del conjunto mientras se mantiene o mejora el rendimiento del modelo de aprendizaje profundo.

5.3.1. Descripción

El funcionamiento de los algoritmos genéticos se basa en los conceptos de **selección natural**, **cruzamiento** y **mutación**. El proceso se puede resumir en los siguientes pasos:

1. **Inicialización**: Se genera una población inicial de posibles soluciones, cada una de ellas representando un subconjunto del conjunto de datos.

2. **Evaluación**: Cada subconjunto de datos (o 'individuo') es evaluado entrenando el modelo con ese subconjunto y midiendo su rendimiento.

3. **Selección**: Se seleccionan los mejores individuos de la población basándose en su rendimiento. Los mejores individuos tienen más probabilidades de ser seleccionados para la siguiente generación.

4. **Cruzamiento**: Se combinan pares de individuos seleccionados para crear nuevos subconjuntos. Esto se realiza intercambiando ejemplos entre los subconjuntos.

5. **Mutación**: Con una pequeña probabilidad, se realizan cambios aleatorios en algunos individuos, como añadir o eliminar ejemplos del subconjunto.

6. **Iteración**: El proceso de evaluación, selección, cruzamiento y mutación se repite durante varias generaciones, con la esperanza de que cada generación produzca soluciones mejores que la anterior.

5.3.2. Aplicación en la reducción de datos

Los **algoritmos genéticos** son especialmente adecuados para la reducción de datos porque permiten explorar un espacio de soluciones muy amplio de manera eficiente. La combinación de individuos y la introducción de mutaciones aleatorias permiten al algoritmo escapar de los óptimos locales, un problema común en la búsqueda local.

El uso de algoritmos genéticos para reducir datos en este contexto implica encontrar subconjuntos de entrenamiento que proporcionen un buen equilibrio entre tamaño y rendimiento. Esto se logra al evaluar diferentes subconjuntos y mejorar las soluciones generación tras generación.

5.3.3. Ventajas y limitaciones

Ventajas: Los algoritmos genéticos son efectivos para explorar grandes espacios de soluciones y tienen una gran capacidad para evitar quedar atrapados en óptimos locales. Son especialmente útiles en problemas donde la solución óptima no es evidente desde el principio.

Limitaciones: Estos algoritmos pueden ser costosos computacionalmente, ya que requieren evaluar muchas soluciones a lo largo de múltiples generaciones. Además, su convergencia a veces puede ser lenta, dependiendo del tamaño del espacio de búsqueda y de los parámetros del algoritmo (tamaño de la población, tasa de mutación, etc.).

5.3.4. Versiones

A lo largo del desarrollo del proyecto, se implementaron diversas versiones del algoritmo genético con el fin de mejorar su rendimiento, adaptabilidad y capacidad para escapar de óptimos locales.

Versión 1 (Genético clásico)

Esta primera versión emplea un operador de cruce simple que intercambia subconjuntos de imágenes seleccionadas entre dos padres. La selección se realiza mediante torneos y se aplica una tasa de mutación fija. El elitismo garantiza la preservación del mejor individuo. Aunque funcional, esta versión presentaba ciertas limitaciones en la calidad de los hijos generados, especialmente en fases avanzadas de la evolución.

Versión 2 (Genético con cruce ponderado)

En esta versión se introdujo un operador de cruce mejorado (*weighted_crossover*), que asigna más imágenes al hijo desde el progenitor con mejor fitness, favoreciendo la herencia de características exitosas. Además, se selecciona solo el mejor de los hijos generados en cada cruce, de esta forma se reduce la propagación de malas soluciones.

Versión 2 libre (ajuste dinámico de tamaño)

Se trata de una variante de la versión 2 donde se permite que el número de imágenes seleccionadas por los hijos varíe dentro de un rango flexible (entre el 50 % y el 150 % del objetivo inicial). Esto se logra mediante el parámetro *adjust_size*, lo que introduce mayor diversidad en la población y fomenta la exploración del espacio de búsqueda.

Versión 3 (Genético con reinicio poblacional)

Esta versión añade un mecanismo de reinicio que se activa si el segundo mejor individuo no mejora en dos generaciones consecutivas. En ese caso, se mantiene el mejor individuo y se reemplaza el resto de la población, generando así nuevos candidatos aleatorios y dando la posibilidad de escapar de estancamientos evolutivos.

5.4. 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.

5.4.1. 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).

En concreto, el algoritmo:

1. Aplica cruce y mutación para generar dos hijos por pareja de padres.
2. Con una cierta probabilidad, aplica búsqueda local a uno de los hijos.
3. Evalúa múltiples vecinos durante un número limitado de evaluaciones (*local_search_evaluations*), seleccionando el mejor encontrado.
4. Incorpora elitismo para asegurar que el mejor individuo no se pierde.

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

5.4.2. 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.

5.4.3. Ventajas y limitaciones

Ventajas:

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

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** [27]: Utilizado en la selección de características y la validación cruzada de modelos. 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.

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

Capítulo 7

Desarrollo Experimental

En este capítulo se exponen los experimentos realizados, los ajustes implementados en los algoritmos 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.

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 [36] 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'.

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.

Las imágenes están organizadas en directorios según su categoría artística:

```
+-- train
|   +-- rock
|   |   +-- image1.jpg
|   |   +-- image2.jpg
|   |   \-- ...
|   +-- paper
|   |   +-- image1.jpg
|   |   +-- image2.jpg
|   |   \-- ...
|   \-- scissors
+-- test (originalmente valid)
|   +-- rock
|   +-- paper
|   \-- scissors
+-- valid (originalmente test)
|   +-- rock
|   +-- paper
|   \-- scissors
```

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".*

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 [37].

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.

Estructura del Dataset

Las imágenes están organizadas en directorios según su categoría artística:

```
+-- Train (originalmente training_set)
|   +-- drawings
|   |   +-- image1.jpg
|   |   +-- image2.jpg
|   |   \-- ...
|   +-- paintings
|   |   +-- image1.jpg
|   |   +-- image2.jpg
|   |   \-- ...
|   +-- sculptures
|   +-- engravings
|   \-- iconography
```

```
+-- Test (originalmente validation_set)
|   +-- drawings
|   +-- paintings
|   +-- sculptures
|   +-- engravings
|   \-- iconography
```

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. MNIST (Modified National Institute of Standards and Technology)

MNIST [40] es un dataset ampliamente utilizado en aprendizaje profundo. Contiene 70,000 imágenes de dígitos escritos a mano, divididas en 60,000 imágenes para el entrenamiento y 10,000 para la prueba.

Estructura del Dataset

Las imágenes tienen un tamaño de 28x28 píxeles y están en escala de grises, con valores de intensidad entre 0 (negro) y 255 (blanco).

Formato de los Datos

Las imágenes están almacenadas en formato IDX, un formato binario específico para este dataset. Se ha realizado una conversión a matrices NumPy para su procesamiento eficiente.

Uso del Dataset

Este conjunto de datos se ha empleado como benchmark para evaluar modelos de clasificación de imágenes, especialmente en arquitecturas convolucionales.

Licencia y uso

El dataset MNIST se distribuye bajo una licencia de dominio público. Fue creado a partir de los datos originales del NIST y está disponible en diversas plataformas, incluyendo la página oficial de Yann LeCun [41].

7.1.4. Comparación con otros datasets

Aqui va una comparación de todos los datasets usados

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, como PAINTING y MNIST. 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 detalladas previamente en el Capítulo 6. Esto permitió comparar algoritmos en condiciones homogéneas, evitando variaciones indeseadas causadas por componentes aleatorios del entorno de ejecución.

En cada prueba, se realizaron cinco ejecuciones paralelas con semillas distintas. Se calcularon las medias de las métricas más relevantes (accuracy, precision, recall y F1-score), junto con el tiempo de ejecución y número de evaluaciones realizadas.

7.3. Evaluación con modelos base

Los primeros experimentos se realizaron con los algoritmos aleatorios, tanto en ResNet50 como en MobileNet, como línea base comparativa. En ambos modelos, se observó una mejora gradual de las métricas al aumentar el porcentaje de datos utilizados. Como era de esperar, ResNet50 ofreció mejores resultados en cuanto a precisión, pero a costa de un mayor tiempo de entrenamiento. MobileNet, en cambio, ofreció una alternativa más rápida y eficiente, aunque con una leve pérdida de rendimiento.

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
aleatorio	100	00:02:12	76,08 %	79,97 %	76,08 %	75,61 %	1

Tabla 7.1: Comparativa de resultados de la generación inicial utilizando el algoritmo **aleatorio** con los modelos **ResNet50** y **MobileNet**.

En las Tabla 7.1 se muestran los resultados obtenidos para diferentes porcentajes iniciales de datos con ambos modelos. Además, en la Figura X se incluye un boxplot donde se comparan los valores de accuracy por porcentaje de datos y por algoritmo.

7.4. Comparativa entre Algoritmos

A lo largo del desarrollo del proyecto, se diseñaron y ajustaron diversas versiones de algoritmos evolutivos y meméticos para seleccionar subconjuntos representativos de imágenes. Esta sección presenta una comparativa entre ellos, no solo desde una perspectiva cuantitativa (accuracy, precisión, etc.), sino también cualitativa, incorporando los aprendizajes progresivos que guiaron su evolución.

7.4.1. Iteración inicial: comparación base

La primera comparativa se centró en tres enfoques principales:

- **Algoritmo aleatorio**, como línea base sin criterio de selección.
- **Búsqueda local**, ajustando iterativamente el conjunto seleccionado.
- **Algoritmo genético (v1)**, combinando subconjuntos mediante operadores clásicos de cruce y mutación.
- **Algoritmo memético**, que combinaba el genético con un proceso de mejora local adicional.

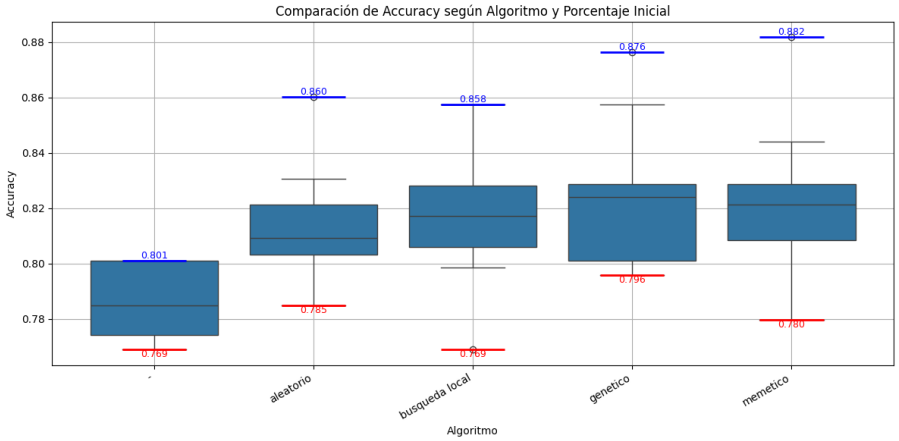

Figura 7.1: Boxplot comparando el *accuracy* alcanzado por cada algoritmo de los iniciales.

El gráfico de la Figura 7.1 ofrece una visión comparativa del rendimiento de los algoritmos evaluados en la etapa inicial del proyecto. Se observa que el algoritmo aleatorio, como era de esperar, presenta una alta dispersión y una mediana de accuracy relativamente baja. 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.

Por su parte, la búsqueda local mejora notablemente la mediana de accuracy, manteniendo una menor variabilidad que el enfoque aleatorio. Esto indica que, pese a su simplicidad, la exploración guiada del espacio de soluciones permite alcanzar resultados más estables y competitivos.

El algoritmo genético (v1) representa un salto adicional en rendimiento. La combinación de operadores evolutivos como la selección, el cruce y la mutación genera soluciones más refinadas, lo que se traduce en una mediana superior y en una mayor compactación de los valores en torno a ella.

Finalmente, el algoritmo memético destaca como el más eficiente dentro de esta primera iteración. Al incorporar una fase de mejora local sobre la evolución genética, logra afinar aún más las soluciones obtenidas. Este enfoque híbrido permite reducir la dispersión y minimizar la probabilidad de obtener soluciones poco efectivas.

7.4.2. Mejoras progresivas: evolución del algoritmo genético

Aunque el algoritmo memético alcanzó el mejor resultado puntual en la comparativa inicial, se decidió continuar el desarrollo sobre el algoritmo genético por varias razones estratégicas.

En primer lugar, el análisis detallado mostró que el genético también presentaba una mediana de accuracy más alta, lo que sugiere un rendimiento más consistente. Además, incluso en sus ejecuciones menos favorables, el genético mostró mejores valores mínimos que el memético, lo cual evidencia una mayor robustez frente a escenarios menos óptimos.

A nivel práctico, el algoritmo genético ofrecía una estructura más simple y modular, lo que facilitó la introducción progresiva de mejoras específicas. Esta simplicidad también implicaba una menor carga computacional al prescindir de la búsqueda local en cada iteración, permitiendo realizar pruebas más rápidas y escalables. Por estos motivos, se consideró que centrar los esfuerzos en afinar el genético aportaría beneficios significativos tanto en eficiencia como en estabilidad, sin renunciar a un rendimiento competitivo.

Por lo tanto, se optó por implementar varias versiones del algoritmo genético, cada una con mejoras específicas:

- **Versión 2 (Genético v2)**: se introdujo un cruce ponderado que daba más peso al progenitor con mejor fitness, y se seleccionaba únicamente el mejor de los dos hijos generado. Esto evitaba la propagación de soluciones poco efectivas y mejoraba la velocidad de convergencia.

- **Versión 3 (Genético v3)**: se añadió una lógica de reinicio poblacional. Si después de varias generaciones no se observaba mejora, se mantenía el mejor individuo y se regeneraban aleatoriamente los demás. Esta estrategia ayudó a salir de óptimos locales sin reiniciar todo el proceso.

Estas versiones se probaron bajo las mismas condiciones que los algoritmos iniciales, lo que permitió compararlos en igualdad de condiciones.

Durante el desarrollo de la Versión 2, se evaluaron distintas estrategias para el operador de mutación. Inicialmente, se aplicaba una regla fija que realizaba un 10 % de permutaciones sobre el subconjunto seleccionado:

$$\text{num\_swaps} = \text{máx}\,(1,\ \text{length} \times 0.1)$$

Sin embargo, esta estrategia podía resultar demasiado conservadora o agresiva, según el tamaño de la solución. En la versión final se adoptó una fórmula más flexible que adapta la magnitud de la mutación en función tanto del subconjunto mutado como del original:

$$\text{num\_swaps} = \text{mín}\,(\text{length} \times 0.15,\ \text{length} \times 0.8)$$

Esta modificación permite un equilibrio más eficaz entre exploración y preservación de estructura, limitando las perturbaciones excesivas sin eliminar la capacidad de escape de óptimos locales.

Debe de comprobarse cómo funciona la fórmula con el ratio añadido:

$$\text{num\_swaps} = \text{mín}\,(\text{length} \times 0.15,\ \text{length} \times \text{ratio} \times 0.8)$$

7.4.3. Evaluación de resultados

A continuación, se presentan dos gráficos en forma de boxplot que comparan el rendimiento de los algoritmos con base en la métrica de *accuracy*. Estas visualizaciones permiten analizar la distribución de resultados, la variabilidad y la consistencia de cada enfoque:

Análisis de los resultados

El primer boxplot (Figura 7.2) muestra que, como era esperable, el *accuracy* mejora progresivamente al incrementar el porcentaje de datos usados. Sin embargo, también se observa que algunos algoritmos, como los genéticos mejorados, obtienen valores muy competitivos incluso con bajos porcentajes iniciales, lo que refuerza su utilidad como técnica de reducción.

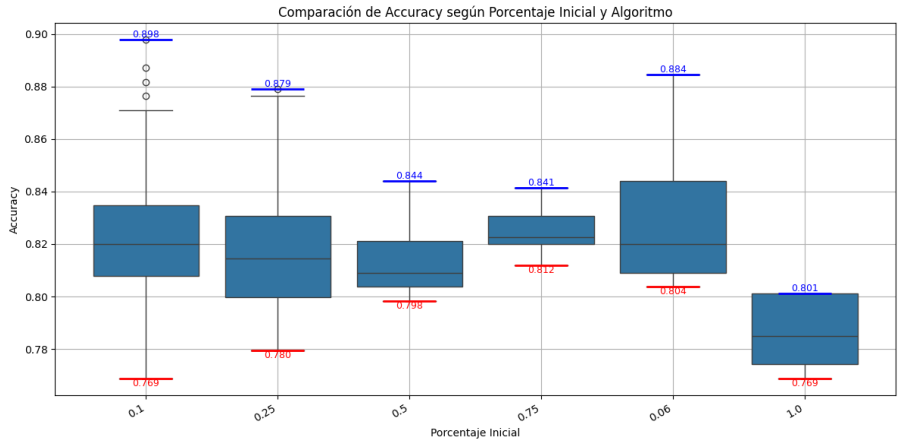

Figura 7.2: Boxplot del *accuracy* alcanzado por los algoritmos en función del porcentaje inicial de imágenes seleccionadas.

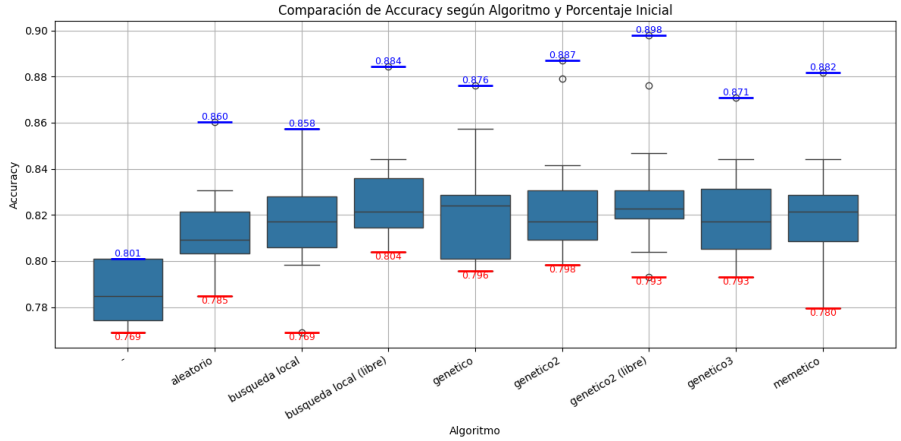

Figura 7.3: Boxplot comparando el *accuracy* alcanzado por cada algoritmo, agrupando los resultados según la estrategia utilizada.

En el segundo boxplot (Figura 7.3), permite comparar directamente el impacto de cada estrategia. Las versiones mejoradas del algoritmo genético (v2 y v3), así como el algoritmo memético, no solo alcanzan medianas de accuracy más altas, sino que también muestran menor variabilidad. Esta menor dispersión indica una mayor estabilidad entre ejecuciones, lo cual es deseable en procesos de optimización con componentes estocásticos.

Además, se observa una clara reducción en los valores atípicos negativos en los algoritmos mejorados, lo que sugiere una menor probabilidad de obtener resultados significativamente bajos. Esto es especialmente relevante en contextos donde se busca fiabilidad en entornos con recursos limitados.

En conjunto, los resultados evidencian que las mejoras introducidas, como el cruce ponderado, la lógica de reinicio poblacional o la combinación con búsqueda local, aportan beneficios sustanciales en términos de rendimiento y robustez. Estos algoritmos superan sistemáticamente a las estrategias más simples (aleatorio y búsqueda local), tanto en precisión como en consistencia, confirmando su idoneidad para tareas de selección de subconjuntos de datos en aprendizaje profundo.

Estas observaciones permiten concluir que las modificaciones introducidas en el algoritmo genético no solo mejoran la media de rendimiento, sino que también aportan consistencia y robustez frente a la aleatoriedad inherente a este tipo de procesos evolutivos.

7.5. Análisis del balance de clases

Adicionalmente, se realizó un análisis del balance de clases en las soluciones finales generadas por cada algoritmo. El objetivo era comprobar si los subconjuntos seleccionados mantenían una representación equilibrada entre las distintas clases del dataset (Paper, Rock y Scissors), o si ciertos algoritmos tendían a favorecer algunas clases frente a otras.

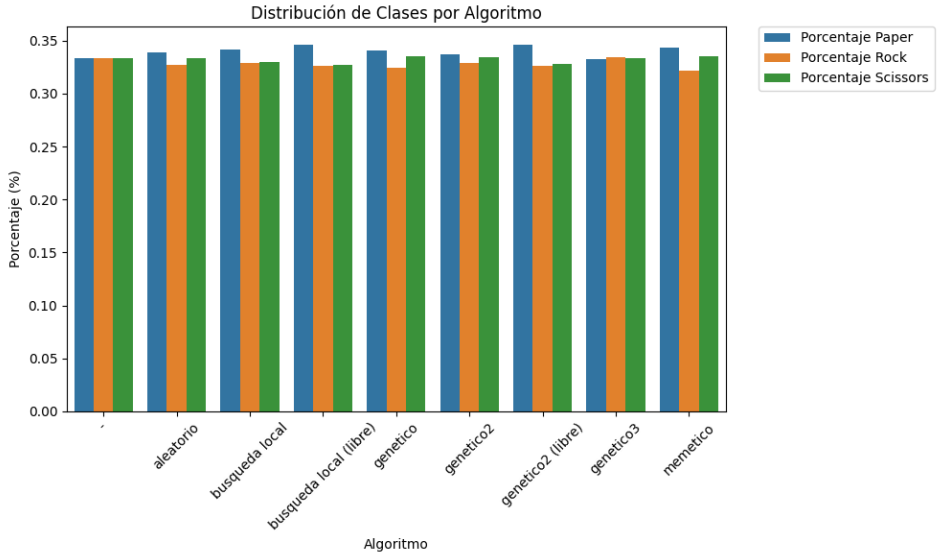

Figura 7.4: Distribución del porcentaje de imágenes por clase en las soluciones generadas por cada algoritmo.

Tal como se observa en la Figura 7.4, los algoritmos más simples como el aleatorio o la búsqueda local presentan ligeras desviaciones, mientras que las versiones más avanzadas, en especial los algoritmos genéticos mejorados y

el memético, logran mantener una distribución más homogénea entre clases. Este equilibrio es clave para evitar sesgos en el modelo entrenado y mejorar métricas como el F1-score.

Aunque si podemos observar una leve tendencia a que los algoritmos necesitan unos pocos datos más de la clase *Paper*, a la vez que reduce el porcentaje de la clase *Rock*.

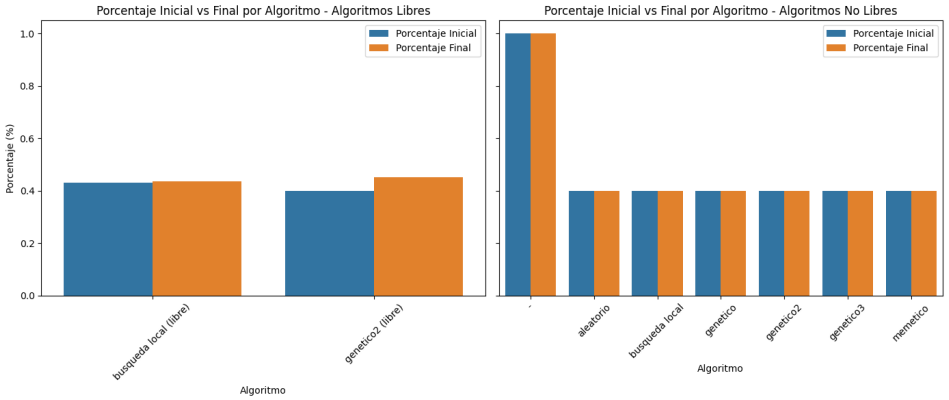

Figura 7.5: Comparación entre porcentaje inicial y final de imágenes seleccionadas por algoritmo.

Por otro lado, se analizó cómo varía el porcentaje de datos seleccionados desde el inicio hasta el final del proceso en cada algoritmo. Este análisis, reflejado en la Figura 7.5, permite observar si los algoritmos tienden a mantener, reducir o incluso aumentar la cantidad de imágenes utilizadas durante su evolución.

Se puede apreciar que los algoritmos con un porcentaje fijo mantienen estable la cantidad de datos seleccionados, como era de esperar debido a su implementación. Sin embargo, en los algoritmos con comportamiento "libre", como *búsqueda local (libre)* o *genético2 (libre)*, si se observa su variabilidad, en particular, ambos algoritmos tienden a aumentar el porcentaje de imágenes seleccionadas respecto al inicial, lo que vuelve a confirmar la tendencia de necesitar más imágenes para mejorar el rendimiento del modelo.

7.6. Validación con el dataset PAINTING

Para comprobar la generalización de los resultados, se repitieron los experimentos con el dataset PAINTING, de mayor tamaño y complejidad.

Falta añadir el análisis detallado de los resultados obtenidos con el dataset PAINTING.

Capítulo 8

Conclusiones

8.0.1. 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 9

Bibliografía

[1] *Ley de Inteligencia Artificial*, es, Page Version ID: 162300059, sep. de 2024. URL: https://es.wikipedia.org/w/index.php?title=Ley_de_Inteligencia_Artificial&oldid=162300059.

[2] *Scrum Guide*. URL: https://scrumguides.org/scrum-guide.html.

[3] *Notion - Gestión de Tareas*, es-es. URL: https://www.notion.com/es-es/help/guides/personal-work-dashboard.

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

[5] *Salario para Data Scientist en España - Salario Medio*, es-es. URL: https://es.talent.com/salary.

[6] *Overview Google Cloud*, es-419-x-mtfrom-en. URL: https://cloud.google.com/docs/overview?hl=es-419.

[7] *What is Cloud Run | Cloud Run Documentation*, en. URL: https://cloud.google.com/run/docs/overview/what-is-cloud-run.

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

[9] P. H. Sydenham y R. Thorn, eds., *Handbook of measuring system design*, en. 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," en, *Artificial Intelligence Review*, vol. 57, n.º 4, pág. 99, mar. de 2024, ISSN: 1573-7462. DOI: 10.1007/s10462-024-10721-6. URL: https://doi.org/10.1007/s10462-024-10721-6.

[11] *ResNet50*, en. URL: https://pytorch.org/hub/nvidia_deeplearningexamples_resnet50/.

[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," en, *Computational Intelligence and Neuroscience*, vol. 2022, n.º 1, pág. 8 777 355, 2022, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2022/8777355, ISSN: 1687-5273. DOI: 10.1155/2022/8777355. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/8777355.

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

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

[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, ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://doi.org/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, ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/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, ISBN: 978-0-472-08460-9. URL: https://books.google.es/books?id=YE5RAAAAMAAJ.

[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), en. Berlin, Heidelberg: Springer, 2012, vol. 379, ISBN: 978-3-642-23246-6 978-3-642-23247-3. DOI: 10.1007/978-3-642-23247-3. URL: http://link.springer.com/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, ISBN: 978-0-201-15767-3. URL: https://books.google.es/books?id=2IIJAAAACAAJ.

[24] J. VanderPlas, *Python Data Science Handbook: Essential Tools for Working with Data*, en. ."O'Reilly Media, Inc.", nov. de 2016, Google-Books-ID: xYmNDQAAQBAJ, ISBN: 978-1-4919-1214-0. URL: 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.

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

[26] *torch.cuda — PyTorch 2.4 documentation*. URL: https://pytorch.org/docs/stable/cuda.html.

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

[28] *NumPy v2.0 Manual*. URL: https://numpy.org/doc/2.0/index.html.

[29] *Polars — Python API reference*. URL: https://docs.pola.rs/api/python/stable/reference/.

[30] *Matplotlib 3.9.3 documentation*. URL: https://matplotlib.org/3.9.3/index.html.

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

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

[33] *Creation of virtual environments*, es. URL: https://docs.python.org/3/library/venv.html.

[34] *Conda Documentation*. URL: https://docs.conda.io/en/latest/.

[35] *cuBLAS Deterministic Algorithms*. URL: https://docs.nvidia.com/cuda/cublas/index.html#cublasApi_reproducibility.

[36] *Rock Paper Scissors Dataset*. URL: https://public.roboflow.com/classification/rock-paper-scissors.

[37] L. Moroney, *Laurence Moroney - The AI Guy*. URL: https://laurencemoroney.com/.

[38] *(Original) Art Images: Drawing/Painting/Sculptures/Engravings*, en. URL: https://www.kaggle.com/datasets/thedownhill/art-images-drawings-painting-sculpture-engraving.

[39] *(Cleaned) Art Images: Drawing/Painting/Sculptures/Engravings*, en. URL: https://www.kaggle.com/datasets/moosecat/art-images-drawings-painting-sculpture-engraving.

[40] *MNIST Dataset*, en. URL: https://www.kaggle.com/datasets/hojjatk/mnist-dataset.

[41] Y. LeCun, *Yann LeCun's Web Page*. URL: http://yann.lecun.com/.