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) [] 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) [] 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, nos encontramos en una era caracterizada por la generación de datos, que crece a un ritmo sin precedentes. Este fenómeno plantea la necesidad de desarrollar métodos capaces de procesar y analizar grandes volúmenes de información de manera eficiente. Los **modelos de aprendizaje profundo** —especialmente las **redes neuronales convolucionales** (CNNs)— han demostrado ser particularmente efectivos en tareas como la **clasificación de imágenes**, el **reconocimiento de patrones** y otras aplicaciones complejas. Sin embargo, el entrenamiento de estos modelos requiere numerosas cantidades de datos, lo que trae consigo importantes desafíos relacionados con el **tiempo de procesamiento**, el **costo económico** y el **consumo de recursos computacionales**.

A medida que los modelos de inteligencia artificial avanzan hacia configuraciones más complejas y precisas, la necesidad de disponer de datos de entrenamiento adecuados y en gran cantidad se hace cada vez más evidente. No obstante, la recolección, almacenamiento y procesamiento de estos datos representan barreras significativas para muchas organizaciones —especialmente aquellas con recursos limitados—, lo que subraya la importancia de explorar enfoques innovadores que permitan la **optimización y reducción** de los conjuntos de datos necesarios para entrenar estos modelos.

1.2. Motivación

La necesidad de **reducir los conjuntos de datos de entrenamiento** surge de la búsqueda por optimizar la **eficiencia** en el desarrollo de modelos de aprendizaje profundo. Si bien las redes neuronales convolucionales

han alcanzado resultados impresionantes, sus **altos costos computacionales** y la extensa cantidad de datos que requieren suponen limitaciones para muchos entornos. Entrenar estos modelos utilizando solo una parte de los datos —seleccionados de manera óptima— podría reducir considerablemente los recursos necesarios (sin comprometer la precisión de los resultados); ello marcaría un avance significativo para la **inteligencia artificial** y sus aplicaciones en entornos con **restricciones de recursos**.

Es aquí donde los **algoritmos meméticos** entran en juego: estos algoritmos combinan las fortalezas de las **técnicas evolutivas** y los **métodos de búsqueda local**, con el fin de optimizar de manera eficiente los conjuntos de datos. La selección de subconjuntos de datos más representativos permite reducir los tiempos de procesamiento y el consumo de recursos sin afectar el rendimiento de los modelos. Así, los algoritmos meméticos se posicionan como una solución prometedora para hacer que el aprendizaje profundo sea más **accesible y eficiente** —incluso en escenarios donde los recursos son limitados—, lo cual es crucial para democratizar el uso de la inteligencia artificial en una amplia gama de aplicaciones.

Este trabajo de fin de grado busca contribuir a esta dirección, explorando la aplicación de algoritmos meméticos en la reducción de datos. La investigación pretende abrir nuevas vías para el desarrollo de modelos más **eficientes**, accesibles y económicamente viables, lo que favorecerá un futuro en el que la inteligencia artificial sea **inclusiva** y más **sostenible**.

1.3. Objetivos

El objetivo principal de este TFG es investigar la aplicación de **algoritmos meméticos** 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 algoritmos meméticos que seleccionen subconjuntos óptimos 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—.

- **Optimizar la eficiencia** del entrenamiento de redes neuronales convolucionales mediante el uso de algoritmos meméticos, 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

La metodología adoptada para el desarrollo de este proyecto ha estado inspirada en el marco ágil **Scrum** [], una estrategia ampliamente utilizada en proyectos que requieren flexibilidad, adaptabilidad y entregas incrementales. Dado que un trabajo de investigación como este demanda constantes ajustes en función de los hallazgos y resultados obtenidos, Scrum ha sido el marco adecuado para organizar el desarrollo de manera iterativa y colaborativa, facilitando la mejora continua durante el proceso.

2.1.1. Metodología Ágil Inspirada en Scrum

El desarrollo del proyecto se organizó en **sprints**, que consistían en ciclos cortos de trabajo (de dos semanas, generalmente), donde se definían objetivos concretos al inicio y se revisaban los avances al final de cada sprint. Esto permitió adaptar el trabajo de acuerdo con los descubrimientos y problemas emergentes a lo largo del proyecto.

Además, el uso de **Git** [] como herramienta para el control de versiones fue clave en la gestión de los avances de desarrollo. Git permitió mantener un registro preciso de cada cambio en el código y los algoritmos, facilitando la colaboración con el tutor y garantizando una trazabilidad eficiente de los ajustes realizados durante cada iteración. Gracias a Git, se pudo revisar, deshacer y modificar el código de forma ágil, integrando nuevos enfoques o mejoras de manera controlada tras cada sprint.

2.1.2. Ciclos de Trabajo

En consonancia con Scrum, cada ciclo de trabajo (o sprint) incluyó una planificación detallada, desarrollo, pruebas y análisis de los resultados obtenidos. Los sprints se estructuraron en torno a objetivos específicos, como

la implementación de un algoritmo o el ajuste de parámetros, y estuvieron formados por las siguientes fases:

- **Planificación del sprint**: Al inicio de cada sprint, se realizaba una sesión de planificación donde se establecían los objetivos específicos del ciclo. Estos objetivos podían incluir tareas como la implementación de un nuevo algoritmo o la optimización de uno existente. La planificación se ajustaba constantemente en función del progreso logrado en sprints anteriores y de los nuevos descubrimientos. Aquí también se establecían las prioridades para asegurar que los esfuerzos se enfocaran en los puntos críticos del proyecto.

- **Desarrollo e implementación de algoritmos**: Durante la fase de desarrollo, se implementaban los algoritmos planificados, como los algoritmos aleatorios, de búsqueda local y genéticos. En lugar de completar el desarrollo de un algoritmo en su totalidad antes de pasar al siguiente, se seguía un enfoque iterativo. Esto permitía construir versiones básicas de los algoritmos que luego se refinaban en iteraciones sucesivas.

  Git fue fundamental en esta etapa, ya que permitía gestionar los cambios en el código de manera efectiva. A medida que se introducían mejoras o correcciones, Git ayudaba a mantener el historial de versiones, facilitando la identificación de regresiones o la comparación entre diferentes enfoques.

- **Pruebas y ajustes iterativos**: Una vez implementados los algoritmos, se realizaban pruebas continuas para evaluar su rendimiento en función de los resultados obtenidos. Esta fase era crucial para identificar posibles áreas de mejora en los algoritmos o en los parámetros utilizados. Los resultados de las pruebas proporcionaban retroalimentación inmediata, lo que permitía realizar ajustes en tiempo real dentro del mismo sprint. Durante este ciclo, los algoritmos se ajustaban y optimizaban iterativamente, lo que permitía identificar errores o ineficiencias de manera temprana.

- **Revisión y análisis de resultados**: Al final de cada sprint, se realizaba una revisión detallada de los resultados obtenidos durante las pruebas. Este análisis incluía no solo la evaluación del rendimiento de los algoritmos, sino también la reflexión sobre los avances logrados respecto a los objetivos iniciales del sprint.

  Durante las reuniones con el tutor, se discutían los resultados y se revisaban posibles cambios en la planificación de los próximos sprints, ajustando las prioridades y estableciendo nuevos objetivos en función de los hallazgos. Esta fase permitía un enfoque incremental, donde las

mejoras y ajustes se incorporaban de manera continua, lo que aseguraba que el proyecto avanzara de forma óptima y se pudieran hacer correcciones oportunas antes de continuar con el siguiente sprint.

- **Documentación de los algoritmos y resultados**: El último paso en cada sprint consistía en documentar los avances alcanzados, tanto en la implementación de los algoritmos como en los resultados obtenidos. Esta documentación no solo reflejaba el estado del proyecto al final de cada sprint, sino que también era esencial para realizar un seguimiento exhaustivo de las mejoras realizadas en cada iteración.

2.1.3. Git como Herramienta de Control de Versiones

Git se utilizó no solo como repositorio de control de versiones, sino también para gestionar de manera eficiente el flujo de trabajo entre el servidor y el entorno local.

Esto permitió:

- **Sincronizar** el código y los datos entre el alumno y el servidor, garantizando que siempre se trabajara con la última versión del proyecto.
- Mantener un **historial de cambios**, facilitando la posibilidad de revertir versiones si era necesario y permitiendo comparaciones entre diferentes iteraciones.
- **Colaborar** de manera eficiente con el tutor, quien podía revisar el código actualizado al final de cada sprint, realizar comentarios y sugerir cambios en cada versión.

2.1.4. Adaptación del Tiempo de los Sprints

Debido a ciertos retrasos en el desarrollo del TFG, fue necesario ajustar la duración de los sprints para asegurarse de que todas las fases y pruebas del proyecto se completaran a tiempo. La flexibilidad de Scrum y el uso de Git para coordinar versiones y tareas pendientes permitieron reorganizar las prioridades y garantizar que los objetivos del proyecto fueran alcanzados sin comprometer la calidad de los resultados.

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** [**?**], 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 iterativos	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** [] de **Compute Engine** [] 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) 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) [**?**] 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**:

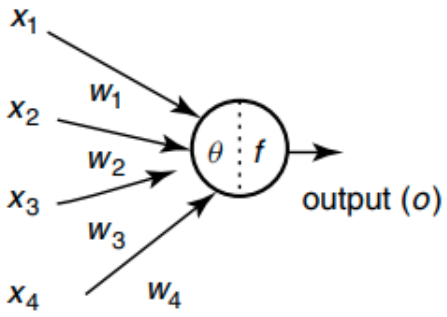

Figura 3.1:

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.

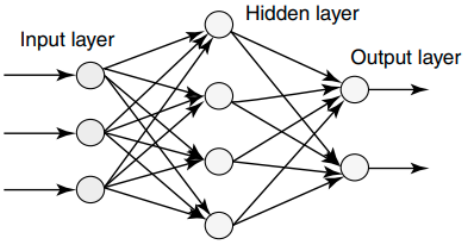

Figura 3.2:

- **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)**: $ReLU(x) = \max(0, x)$, 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, CNNs) 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 CNNs 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 CNNs 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

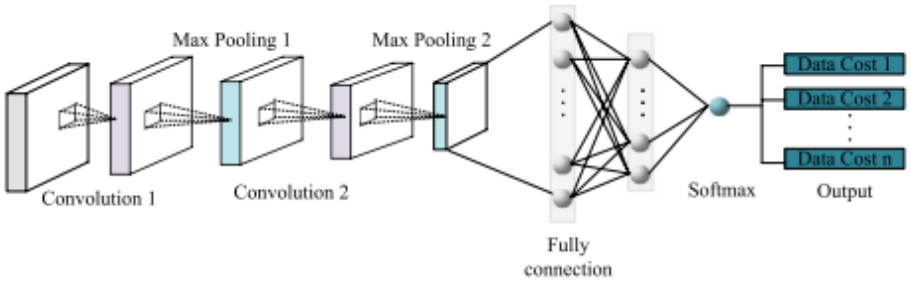

Figura 3.3: En esta imagen extraída de [**?**] puede observarse de la estructura de una red neuronale convolucional. Junto a sus capas convolucionales.

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. Datasets

En el aprendizaje profundo, los datasets son colecciones de datos etiquetados o no etiquetados que se utilizan para entrenar modelos. Estos conjuntos de datos suelen contener 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.

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

Rock, Paper, Scissors [**?**] fue creado originalmente por Laurence Moroney y se utiliza para clasificar imágenes de las manos representando los gestos de 'piedra', 'papel' y 'tijeras'. El conjunto de datos contiene alrededor de 2,500 imágenes distribuidas en tres categorías: piedra, papel y tijeras. Las imágenes están en color y tienen un tamaño de 300x300 píxeles.

En este trabajo, se ha utilizado el dataset de **Rock, Paper, Scissors** para evaluar el rendimiento del modelo en un problema de clasificación de imágenes más variado y natural, que involucra múltiples clases. Además, permite explorar la eficacia de los algoritmos meméticos en un entorno más cercano al reconocimiento de objetos, lo que añade mayor complejidad al problema.

3.4.2. MNIST (Modified National Institute of Standards and Technology)

MNIST [**?**] es uno de los datasets más utilizados en el campo del aprendizaje automático y el aprendizaje profundo. Contiene 70,000 imágenes de dígitos escritos a mano (60,000 para el conjunto de entrenamiento y 10,000 para el de prueba). Las imágenes son en escala de grises y tienen un tamaño de 28x28 píxeles, con cada píxel representando una intensidad de color entre 0 (negro) y 255 (blanco).

Este dataset se utiliza comúnmente como **benchmark** para evaluar modelos de clasificación de imágenes, particularmente en arquitecturas convolucionales.

La simplicidad de **MNIST** lo hace ideal para probar modelos de redes neuronales convolucionales, ya que ofrece un equilibrio entre un problema

fácil de entender, pero con suficiente complejidad para que los modelos más avanzados demuestren mejoras.

3.4.3. Comparación con otros datasets

La selección de estos dos datasets responde a la necesidad de evaluar los algoritmos meméticos en distintos niveles de complejidad. **MNIST**, con imágenes en escala de grises de bajo nivel de complejidad, proporciona una referencia clara y estandarizada para comparar el rendimiento y la reducción de datos. Por otro lado, el dataset de **Rock, Paper, Scissors** introduce más desafíos visuales y complejidades, permitiendo analizar cómo los algoritmos meméticos se comportan en escenarios más complejos que podrían ser representativos de aplicaciones más reales en visión por computadora.

3.5. 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.

3.5.1. ResNet50

ResNet50 [**?**] es una arquitectura de red neuronal convolucional introducida por Kaiming He et al. en 2015. 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 está compuesta por 50 capas, de las cuales 49 son capas convolucionales y una es una capa totalmente conectada. La arquitectura incluye

capas de normalización y activación (ReLU), así como una estructura que permite saltos de conexión, lo que significa que la entrada de una capa se suma a la salida de una capa posterior.

3.5.2. MobileNetV2

MobileNet [] 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 [], 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 [**?**, **?**]. 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** [**?**, **?**].

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** [**?**, **?**] permiten reducir el tamaño de los conjuntos de datos manteniendo una **diversidad** adecuada, lo cual es crucial para evitar el **sobreajuste** 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** [**?**, **?**].

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 [**?**, **?**, **?**, **?**]. 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 [**?**, **?**]. 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 [**?**].

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** [**?**, **?**].

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** y el **algoritmo genético**

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.).

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 de carpetas y archivos del proyecto se organizó para facilitar el acceso modular al código, datos y documentación. Cada componente se divide en módulos específicos que agrupan funcionalidad relacionada, lo cual permite una mejor organización, mantenibilidad y pruebas independientes.

6.1.1. Estructura de Archivos

La estructura del proyecto es la siguiente:

- `data` – Almacena los dataset utilizados por el proyecto.
- `docs` – Documentación del proyecto en latex.
- `img` – Imágenes generadas en el proyecto.
- `LICENSE` – Términos de distribución del proyecto.
- `logs` – Registros para el control y seguimiento de las evaluaciones.
- `README.md` – Descripción general.
- `requirements.txt` – Dependencias del proyecto.
- `results` – Resultados generados por el proyecto (fitness, tiempos, etc.).
    - `csvs` – Resultados de las ejecuciones guardados en tablas.

    - `salidas` – Resultados devueltos por pantalla por el programa.

- `scripts` – Scripts y programas secundarios para ejecutarse en el servidor GPU.
- `src` – Código fuente del proyecto.
- `tmp` – Archivos temporales.
- `utils` – Módulos y scripts de utilidad.

6.2. Herramientas y Lenguajes de Programación

El desarrollo del proyecto se ha llevado a cabo utilizando **Python 3.10** [] 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** []: 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** []: 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** []: Para operaciones matemáticas y manipulación de matrices, siendo una herramienta esencial en el procesamiento de datos.
- **Polars 1.9.0** []: Biblioteca para manejar DataFrames de gran tamaño, elegida por su rendimiento superior en comparación con Pandas.
- **matplotlib 3.9.2** []: Biblioteca utilizada para la generación y visualización de gráficas.

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` [**?**]. 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` [**?**] 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. Módulo Principal

El módulo `main.py` constituye el núcleo de la implementación de los algoritmos de optimización aplicados en el proyecto. Este archivo contiene el código necesario para inicializar, configurar, y ejecutar los algoritmos sobre un conjunto de datos, además de registrar y visualizar los resultados de las evaluaciones de rendimiento. A continuación, se describen las secciones principales del módulo.

Para asegurar que los experimentos sean reproducibles, se implementa la función `set_seed(seed)`, que toma un valor de semilla y lo establece para todas las librerías de manejo aleatorio utilizadas (incluyendo torch, numpy, y random). Esto permite obtener resultados consistentes en cada ejecución del mismo experimento.

La función main es el núcleo del archivo y permite ejecutar uno de varios algoritmos según los parámetros especificados. Toma los siguientes parámetros principales:

- `initial_percentage`: Porcentaje inicial de datos a evaluar.
- `max_evaluations` y `max_evaluations_without_improvement`: Límite de evaluaciones para el algoritmo y cantidad máxima sin mejora antes de detenerse.
- `algoritmo`, `metric`, y `model_name`: Especifican el tipo de algoritmo, la métrica de evaluación y el modelo que se usa.

Los pasos de la función principal de `main` 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.

6.4.3. Módulos de Iniciación

El módulo inicial de Iniciación fue el de `main.py`, ya que establece un task_id —que se utiliza para identificar la tarea— y unos valores para los parámetros principales del `main`.

El resto de estos módulos (`generator_initial.py` y `generator.py`) son una extensión de `main.py`, por el hecho de que están diseñados para realizar

ejecuciones comparativas de diferentes algoritmos de optimización sobre una serie de porcentajes iniciales de datos de entrada, evaluando el rendimiento de cada algoritmo.

Estos módulos también 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. **Optimización 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.

Capítulo 7

Desarrollo Experimental

Tras la explicación del contenido necesario para el entendendimiento y desarrollo del TFG en los anteriores capítulos, en este capítulo se prcedera a explicar las pruebas y posteriores mejoras que se han realizado y los posteriores resultados obtenidos.

Las pruebas iniciales que se plantearon fueron tomar un dataset simple para realizar las primeras pruedas, y ya cuando funcionase correctamente, probar con otro dataset más complejo o realista. Para ello se decidió usar el dataset de **RPS**.

Para obtener unos primeros resultados con este dataset, se planteó usar el modelo de **Resnet50**. Se hicieron unas pruebas con distintos porcentajes para ver distintos resultados posibles.

Algoritmo	**Porcentaje Inicial**	**Duracion**	**Accuracy (Avg)**	**Precision (Avg)**	**Recall (Avg)**	**F1-score (Avg)**	**Evaluaciones Realizadas**
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

Tabla 7.1: Resultados de la generación inicial con **Resnet50**

Fijandonos en la tabla 7.1, las duraciones que se indican son lo que tarda en ejecutarse todas las evaluaciones que se indican. Vemos que son tiempos muy grandes, por ello se decidió probar el algoritmo **MobileNet** para agilizar el proceso.

Algoritmo	Porcentaje Inicial	Duracion	Accuracy (Avg)	Precision (Avg)	Recall (Avg)	F1-score (Avg)	Evaluaciones Realizadas
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 7.2: Resultados de la generación inicial con **MobileNet**

Capítulo 8

Conclusiones