COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

Grado en Ingeniería Matemática e Inteligencia Artificial

Trabajo Fin de Grado

Aplicación de la Inteligencia Artificial en la Compresión de Datos

Autor: José Andrés Ridruejo Tuñón

Director: Emanuel Gastón Mompó Pavesi

Madrid, 22 de Agosto

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Aplicación de la Inteligencia Artificial en la Compresión de Datos en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2024/25 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: **José Andrés Ridruejo Tuñón** Fecha: 22/08/2025

Autorizada la entrega del proyecto

El Director del Proyecto

Fdo.: **Emanuel Gastón Mompó Pavesi** Fecha: 22/08/2025

MOMPO PAVESI EMANUEL GASTON - 21002240C

Signed by: MOMPO PAVESI EMANUEL GASTON - 21002240C

Date: 2025-08-28 21:18:02 CEST

AGRADECIMIENTOS

Universidad Pontificia Comillas
Escuela Técnica Superior de Ingeniería (ICAI)
Grado en Ingeniería Matemática e Inteligencia Artificial

Application of Artificial Intelligence in Data Compression

Author: José Andrés Ridruejo
Supervisor: Emanuel Gastón Mompó Pavesi
Collaborating Institution: Universidad Pontificia Comillas – ICAI

Abstract

This project studies data compression from information theory to its application through artificial intelligence models. Classical compression algorithms such as Huffman and LZ77 for lossless compression and DCT for lossy compression were implemented in Python. Subsequently, different neural network architectures applied to image compression were explored, ultimately developing a compact variational autoencoder called *yosemite_model*. The objective was to overtrain this model on a reduced dataset, achieving nearly perfect reconstructions and a final model smaller than the original dataset, thus obtaining specific compression.
Keywords: Autoencoder, VAE, image compression, overfitting, LoRA, PEFT.

1. Introduction

Data compression is a fundamental area of computer science and telecommunications. In general terms, to compress means to understand: in order to represent information more efficiently it is necessary to capture its internal regularities. After reviewing fundamental concepts such as entropy and Kolmogorov complexity, traditional compression algorithms (Huffman and LZ77) were implemented in Python and, subsequently, different neural network architectures for image compression were evaluated.

The main goal was to test the capacity of a very small model to overfit a set of 44 images of 512×512 pixels and achieve overfitting on those images. This idea is based on the universal approximation theorem, which states that neural networks with a certain structure can, in principle, approximate any continuous function with any desired degree of accuracy; in this case an image is interpreted as the output of an unknown function that can be approximated by a neural network.

The base model selected, named *yosemite_model*, was optimized by applying different parameter-reduction and storage techniques. Its results were compared with different variants, and it was concluded that the final design with separable convolutions and global average pooling offered the best trade-off between reconstruction quality and model compression.

2. Project Definition

The project was carried out in two phases.

First, classical compression algorithms were studied and implemented in Python:

- **Huffman**, which assigns shorter codes to the most frequent symbols using a binary tree.
- **LZ77**, which replaces repeated sequences with references to earlier positions in the data.
- **DCT (Discrete Cosine Transform)**, which transforms image blocks into the frequency domain, concentrating energy in a few coefficients, the basis of JPEG.

Second, different artificial intelligence models applied to compression were explored:

- A **convolutional neural network (CNN)** as a first approach for images.
- A **recurrent neural network (RNN)**, oriented towards text sequences.

Finally, a **compact variational autoencoder (*yosemite_model*)** was chosen, designed to overtrain on a small dataset and obtain nearly perfect reconstructions with a model smaller than the dataset itself.

3. Description of the Model/System/Tool

The *yosemite_model* consists of:

Different techniques were investigated to reduce the number of parameters and the size of the model: channel reduction in each layer, elimination of the dense bottleneck via global average pooling or 1×1 convolutions, use of depthwise/separable convolutions, increased intermediate downsampling, quantization in INT8 or FP16 formats, weight pruning, matrix factorization and parameter sharing, use of pretrained latent tables, and reduced precision of activations.

4. Results

Model	Training	MSE	PSNR (dB)	Parameters	Disk Size	Compression
Original VAE	10 min	512.7	21.0	18.2 M	69.6 MB	0.50
Reduced VAE (fewer channels)	19 min	893.4	18.6	4.5 M	17.0 MB	2.03
Optimized VAE (sep. conv + GAP)	17 min	742.9	19.4	4.4 M	16.9 MB	2.04
Optimized VAE + quantization	18 min	689.2	19.8	4.4 M	4.2 MB	8.21
Optimized VAE + pruning + quantization	20 min	523.6	20.9	4.4 M (50 % zeros)	2.1 MB (sparse)	16.43

Tabla 1: Comparison between the original VAE and the optimized variants

Among all optimizations, the most effective were separable convolutions, global average pooling, quantization, and pruning. The dataset used consisted of 44 RGB images of 512×512 pixels (33 MB). The final model (4 MB) was smaller than the dataset itself, fulfilling the compression objective.

5. Conclusions

It has been shown that a **compact autoencoder** can act as a specific compressor for a reduced set of images, taking up less space than the original dataset it is overtrained on. This approach is based on the universal approximation theorem and directly connects with the theoretical limits of compression.

Although multiple optimizations were explored, the most effective were **separable convolutions**, **global average pooling**, **quantization**, and **pruning**.

Future Research

A promising line of work consists of training a **large model** on a varied set of images to learn general representations, and then applying **light fine-tuning** with techniques such as LoRA on small datasets. This way, a single base model would be stored, and each new image set could be compressed by adding very few extra parameters at low computational cost, resulting in greater efficiency in the long term.

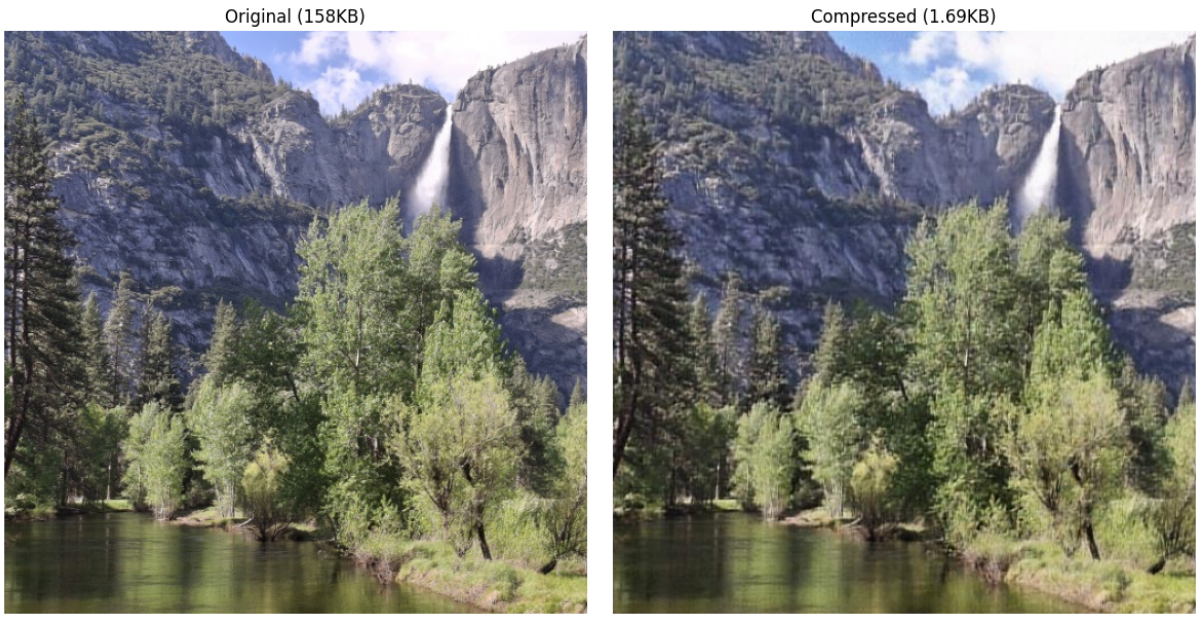


Figura 1: Illustrative example of the result: high-fidelity reconstructions.

Referencias

[1] M. Mahoney, *Data Compression Explained*, 2010. Available at: https://mattmahoney.net/dc/dce.html

[2] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, N. Johnston, *Variational Image Compression with a Scale Hyperprior*, 2018. Available at: https://arxiv.org/pdf/1802.01436v2

[3] A. X. et al., *LLMZip: Lossless Text Compression using Large Language Models*, 2023. Available at: https://arxiv.org/pdf/2306.04050v2

[4] T. van Rozendaal, I. A. M. Huijben, T. S. Cohen, *Overfitting for Fun and Profit: Instance-Adaptive Data Compression*, 2021. Available at: https://arxiv.org/pdf/2101.08687

[5] M. Hutter, *500,000€ Prize for Compressing Human Knowledge*, 2006. Available at: http://prize.hutter1.net/index.htm

[6] F. Bellard, *NNCP v2: Lossless Data Compression with Transformer*, 2023. Available at: https://bellard.org/nncp/nncp_v2.pdf

Universidad Pontificia Comillas
Escuela Técnica Superior de Ingeniería (ICAI)
Grado en Ingeniería Matemática e Inteligencia Artificial

Aplicación de la Inteligencia Artificial en la Compresión de Datos

Autor: José Andrés Ridruejo
Director: Emanuel Gastón Mompó Pavesi
Entidad Colaboradora: Universidad Pontificia Comillas – ICAI

Resumen

Este proyecto estudia la compresión de datos desde la teoría de la información hasta su aplicación mediante modelos de inteligencia artificial. Se implementaron en Python algoritmos clásicos de compresión como Huffman y LZ77 para compresión sin pérdidas y DCT para compresión con pérdidas. Posteriormente, se exploraron diferentes arquitecturas de redes neuronales aplicadas a la compresión de imágenes, desarrollando finalmente un autoencoder variacional compacto denominado *yosemite_model*. El objetivo fue sobreentrenar este modelo en un dataset reducido, logrando reconstrucciones casi perfectas y un modelo final más pequeño que el conjunto original, obteniendo así compresión específica.
Palabras clave: Autoencoder, VAE, compresión de imágenes, overfitting, LoRA, PEFT.

1. Introducción

La compresión de datos es un área fundamental de la informática y las telecomunicaciones. En términos generales, comprimir significa comprender: para representar la información de forma más eficiente es necesario capturar sus regularidades internas. Tras una revisión de conceptos fundamentales como entropía y complejidad de Kolmogórov, se han implementado en Python algoritmos de compresión tradicionales (Huffman y LZ77) y, posteriormente, se han evaluado distintas arquitecturas de redes neuronales para la compresión de imágenes.

El objetivo principal ha sido comprobar la capacidad de un modelo muy pequeño para sobreentrenarse sobre un banco de 44 imágenes de 512×512 píxeles y hacer overfitting en dichas imágenes. Esta idea se sustenta en el teorema de aproximación universal, el cual afirma que las redes neuronales con una determinada estructura pueden, en principio, aproximarse a cualquier función continua con cualquier grado deseado de precisión; en este caso se está interpretando una imagen como el resultado de una función desconocida que puede ser aproximada por una red neuronal.

El modelo base seleccionado, denominado *yosemite_model*, se optimizó aplicando diferentes técnicas de reducción de parámetros y almacenamiento. Se compararon sus resultados con las distintas variantes, y se concluyó que el diseño final con convoluciones separables y global average pooling fue el que ofreció la mejor relación entre calidad de reconstrucción y compresión del modelo.

2. Definición del proyecto

El proyecto se planteó en dos fases.

En primer lugar, se estudiaron e implementaron algoritmos clásicos de compresión en Python:

- **Huffman**, que asigna códigos más cortos a los símbolos más frecuentes mediante un árbol binario.
- **LZ77**, que sustituye secuencias repetidas por referencias a posiciones anteriores en los datos.
- **DCT (Discrete Cosine Transform)**, que transforma bloques de la imagen al dominio de la frecuencia, concentrando la energía en pocos coeficientes, técnica base de JPEG.

En segundo lugar, se exploraron diferentes modelos de inteligencia artificial aplicados a compresión:

- Una **red convolucional (CNN)**, como primer enfoque para imágenes.
- Una **red recurrente (RNN)**, orientada a secuencias de texto.

Finalmente, se optó por un **autoencoder variacional compacto (*yosemite_model*)**, diseñado para sobreentrenarse en un dataset pequeño y obtener reconstrucciones casi perfectas con un modelo más pequeño que el propio conjunto de datos.

3. Descripción del modelo/sistema/herramienta

El *yosemite_model* se compone de:

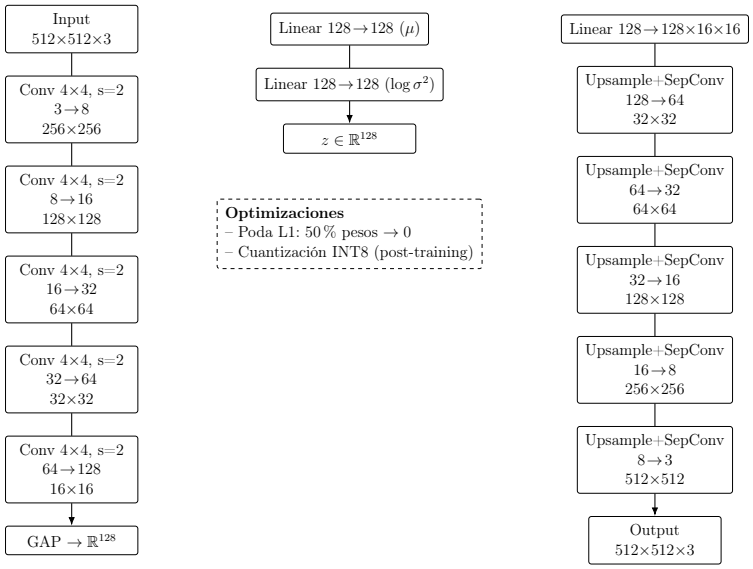


Figura 2: Esquema de la arquitectura del modelo autoencoder

Se investigaron distintas técnicas para reducir el número de parámetros y el tamaño del modelo: la reducción de canales en cada capa, la eliminación del *bottleneck* denso mediante

global average pooling o convoluciones 1×1, el uso de convoluciones depthwise/separables, un mayor downsampling intermedio, la cuantización en formatos INT8 o FP16, la poda de pesos, la factorización de matrices y la compartición de parámetros, el uso de tablas de latentes preaprendidos y la reducción de la precisión de activaciones.

4. Resultados

Modelo	Entrenamiento	MSE	PSNR (dB)	Parámetros	Tamaño en disco	Compresión
VAE original	10 min	512.7	21.0	18.2 M	69.6 MB	0.50
VAE reducido (canales menores)	19 min	893.4	18.6	4.5 M	17.0 MB	2.03
VAE optimizado (sep. conv + GAP)	17 min	742.9	19.4	4.4 M	16.9 MB	2.04
VAE optimizado + cuantización	18 min	689.2	19.8	4.4 M	4.2 MB	8.21
VAE optimizado + poda + cuantización	20 min	523.6	20.9	4.4 M (50 % ceros)	2.1 MB (sparse)	16.43

Tabla 2: Comparación entre el VAE original y las variantes optimizadas

De todas las optimizaciones, las más eficaces fueron las convoluciones separables, el global average pooling, la cuantización y la poda. El dataset utilizado fue de 44 imágenes RGB de 512×512 píxeles (33 MB). El modelo final (4 MB) resultó más pequeño que el propio conjunto de datos, cumpliendo el objetivo de compresión.

5. Conclusiones

Se ha demostrado que un **autoencoder compacto** puede actuar como un compresor específico para un conjunto reducido de imágenes, ocupando menos espacio que el dataset original al que se sobreentrena. Esta aproximación está fundamentada en el teorema de aproximación universal y conecta directamente con los límites teóricos de la compresión.

Aunque se exploraron múltiples optimizaciones, las más efectivas fueron las **convoluciones separables**, el **global average pooling**, la **cuantización** y la **poda**.

Investigación futura

Una línea prometedora consiste en entrenar un **modelo grande** sobre un conjunto variado de imágenes para aprender representaciones generales, y posteriormente aplicar **fine-tuning ligero** con técnicas como LoRA en datasets pequeños. Con ello se almacenaría un único modelo base y cada nuevo conjunto de imágenes podría comprimirse añadiendo muy pocos parámetros extra y con bajo coste computacional, resultando más eficiente a largo plazo.

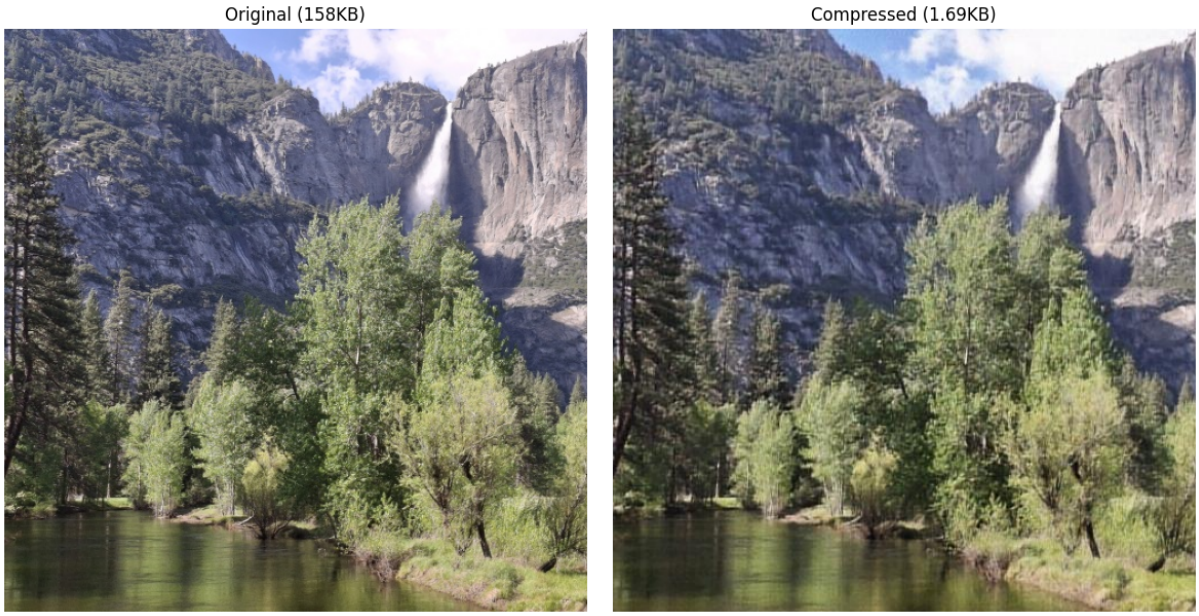


Figura 3: Ejemplo ilustrativo del resultado: reconstrucciones de alta fidelidad.

Referencias

[1] M. Mahoney, *Data Compression Explained*, 2010. Disponible en: https://mattmahoney.net/dc/dce.html

[2] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, N. Johnston, *Variational Image Compression with a Scale Hyperprior*, 2018. Disponible en: https://arxiv.org/pdf/1802.01436v2

[3] A. X. et al., *LLMZip: Lossless Text Compression using Large Language Models*, 2023. Disponible en: https://arxiv.org/pdf/2306.04050v2

[4] T. van Rozendaal, I. A. M. Huijben, T. S. Cohen, *Overfitting for Fun and Profit: Instance-Adaptive Data Compression*, 2021. Disponible en: https://arxiv.org/pdf/2101.08687

[5] M. Hutter, *500,000€ Prize for Compressing Human Knowledge*, 2006. Disponible en: http://prize.hutter1.net/index.htm

[6] F. Bellard, *NNCP v2: Lossless Data Compression with Transformer*, 2023. Disponible en: https://bellard.org/nncp/nncp_v2.pdf

Índice

Universidad Pontificia Comillas
Escuela Técnica Superior de Ingeniería (ICAI)
Grado en Ingeniería Matemática e Inteligencia Artificial

Aplicación de la Inteligencia Artificial en la Compresión de Datos

Autor: José Andrés Ridruejo
Director: Emanuel Gastón Mompó Pavesi
Entidad Colaboradora: Universidad Pontificia Comillas – ICAI

Resumen

Este proyecto estudia la compresión de datos desde la teoría de la información hasta su aplicación mediante modelos de inteligencia artificial. Se implementaron en Python algoritmos clásicos de compresión como Huffman y LZ77 para compresión sin pérdidas y DCT para compresión con pérdidas. Posteriormente, se exploraron diferentes arquitecturas de redes neuronales aplicadas a la compresión de imágenes, desarrollando finalmente un autoencoder variacional compacto denominado *yosemite_model*. El objetivo fue sobreentrenar este modelo en un dataset reducido, logrando reconstrucciones casi perfectas y un modelo final más pequeño que el conjunto original, obteniendo así compresión específica.

Palabras clave: Autoencoder, VAE, compresión de imágenes, overfitting, LoRA, PEFT.

1. Introducción

1.1. Contexto y Motivación

La compresión de datos es fundamental en la informática moderna; se necesita para optimizar el almacenamiento, reducir el ancho de banda en la transmisión de video como en plataformas de streaming y mejorar la eficiencia en el procesamiento de datos. Desde los primeros ordenadores, la capacidad de eliminar la redundancia de la información ha sido un motor clave para la innovación tecnológica. Sin embargo, los algoritmos clásicos, que han servido durante años ayudando a reducir el tamaño de los datos, llevan décadas sin tener cambios fundamentales . Esta falta de desarrollo contrasta con el avance exponencial de la inteligencia artificial (IA).

El surgimiento de las redes neuronales profundas ha revolucionado campos como la superresolución de imágenes y la predicción del lenguaje. El problema de la compresión de datos es, en esencia, un problema de modelado y codificación. Mientras que la codificación es un problema que ya está resuelto en gran medida, el modelado —el proceso de identificar patrones y estructuras en los datos para describirlos de manera más concisa— sigue siendo un reto abierto [1] . La IA, con su capacidad para el reconocimiento de patrones, es la candidata natural para abordar este problema, ofreciendo la posibilidad de grandes mejoras en la forma en que comprendemos y comprimimos la información. Este trabajo de fin de grado se sitúa en la intersección de estas dos disciplinas, explorando el potencial de la IA para llevar la compresión de datos a un nuevo límite.

1.2. La Compresión como un Camino hacia la Inteligencia Artificial General (AGI)

El hilo conductor de este trabajo es la tesis de que la capacidad de comprimir bien un conjunto de datos está intrínsecamente ligada a la inteligencia. Esta idea, que reduce un concepto como la inteligencia a una métrica tangible, el tamaño de un archivo, es la base de Hutter Prize [5], una competición que busca comprimir todo el conocimiento humano para acercarnos a una AGI. Este concurso, que ofrece un premio de 500.000€, desafía a los participantes a comprimir el archivo enwik9 (un extracto de 1 GB de Wikipedia) a un tamaño menor que el récord actual (9.31).

Para lograr la máxima compresión, un programa debe encontrar y explotar las regularidades y patrones subyacentes en el texto. Esto es un acto de modelado, y un compresor que supera a sus predecesores debe ser, por definición, más inteligente en su capacidad de modelar el conocimiento humano.

Esta idea tiene un fundamento matemático en la Complejidad de Kolmogorov, la cual define la complejidad de un objeto como la longitud de su programa más corto que puede generarlo. Desde esta perspectiva, la compresión es la búsqueda de la descripción más concisa de un conjunto de datos, lo que implica una comprensión profunda de su estructura.

La AGI, por lo tanto, podría ser vista como un compresor universal hipotético, capaz de comprender el mundo con la máxima eficiencia, alcanzando el límite teórico de la compresión de cualquier fuente de datos.

1.3. Objetivos

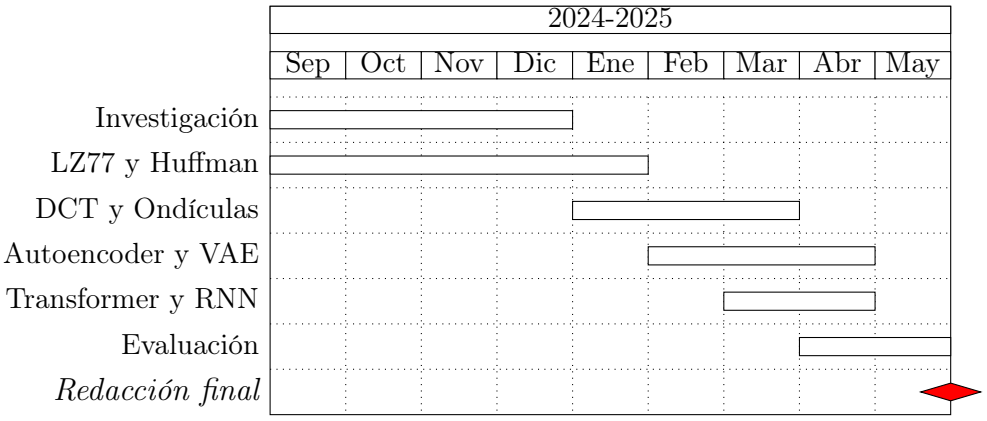
Este trabajo tiene como objetivo principal analizar diferentes técnicas de compresión, comenzando por implementar los algoritmos clásicos utilizados en formatos de compresión tanto lossy como lossless. Posteriormente, se probarán modelos de aprendizaje profundo que puedan competir contra los algoritmos tradicionales.

Para lograr estos fines, se han definido una serie de objetivos concretos:

- **Implementación y estudio de algoritmos clásicos:** Se abordará la implementación de algoritmos tradicionales de compresión sin pérdida, como Huffman y LZ77, y técnicas con pérdida basadas en transformadas, como la Transformada de Coseno Discreta (DCT).
- **Diseño e implementación de modelos basados en IA:** Se desarrollarán modelos de red neuronal, en particular un Autoencoder Variacional (VAE), para la compresión con pérdida de imágenes. Además, se explorará la aplicación de arquitecturas avanzadas como los Transformers para la compresión de texto.
- **Análisis y comparación:** Realizar un análisis comparativo de los modelos en términos de número de parámetros, tamaño en disco y error de reconstrucción para evaluar la eficiencia de la "compresión por overfitting".
- **Investigación de vanguardia:** Proponer una línea de investigación futura basada en la técnica de Low-Rank Adaptation (LoRA) para escalar la idea de la "compresión por memorización" a múltiples datasets de manera eficiente.

1.4. Planificación

Se siguió en gran medida el diagrama de Gantt propuesto en el anexo B, que fue el siguiente:



El desarrollo de este proyecto se estructuró en varias fases: una etapa inicial de estudio de la teoría de información, entropía y conceptos fundamentales de la compresión de información; la fase experimental, que incluyó la implementación y el entrenamiento de los modelos de autoencoder y VAE. Aquí la investigación se prolongó hasta junio ya que el VAE necesitaba de varias optimizaciones. En julio se trató de implementar una Red neuronal recurrente y un transformer, pero no se consiguió el resultado deseado ya que son arquitecturas demasiado pesadas para el dataset. Finalmente se hizo un análisis de los resultados obtenidos y las métricas de rendimiento; y, finalmente. la redacción del informe que sintetiza todos los resultados.

1.5. Viabilidad Económica

Dado el alto coste computacional y económico de entrenar modelos de IA a gran escala, es necesario desarrollar alternativas más eficientes. Este trabajo analiza cómo las arquitecturas neuronales de menor tamaño y las técnicas de optimización de parámetros pueden ser soluciones asequibles, con el fin de facilitar un mayor acceso a la tecnología de la IA.

1.6. Estructura del Trabajo

Este informe comienza con una revisión del estado del arte en algoritmos de compresión y modelos de inteligencia artificial, como los autoencoders. A continuación, la metodología detalla la arquitectura y el entrenamiento de los modelos. Posteriormente, se presentan y analizan los resultados de los experimentos. El documento concluye con los hallazgos principales y una propuesta de investigación futura.

2. Fundamentos de la Teoría y Práctica de la Compresión de Datos

2.1. ¿Qué es la Compresión de Información?

La compresión de información es el proceso de codificar datos utilizando menos bits de los que se usarían en su representación original. Este proceso se basa en la eliminación de la redundancia, que se refiere a la información predecible, repetitiva o no esencial dentro de un conjunto de datos. En términos técnicos, un compresor opera transformando una cadena de símbolos de entrada en una cadena de bits de salida más corta, con la condición de que la información original pueda ser reconstruida total o parcialmente.

Existen dos procesos fundamentales en la compresión: el **modelado** y la **codificación**. El modelado es la parte más compleja, ya que implica construir un modelo estadístico de la fuente de datos para predecir la probabilidad de que ocurra cada símbolo. Un modelo eficaz reconoce patrones y dependencias. La codificación, por su parte, consiste en asignar códigos más cortos a los símbolos más probables, basándose en la información proporcionada por el modelo.

2.2. La Navaja de Ockham: Comprimir es Comprender

Más allá de su aplicación técnica, la compresión de datos está estrechamente ligada al concepto de comprensión. Un compresor obtiene una alta tasa de compresión porque identifica una estructura o un modelo en los datos, lo que permite representarlos de forma más compacta.

El *Hutter Prize* muestra esta idea de forma clara: mide la inteligencia según lo pequeño que quede un archivo al comprimirlo. La lógica es que un compresor "inteligente" es aquel que mejor entiende el contenido del corpus `enwik9`. Esto se relaciona con el principio de *Minimum Description Length (MDL)*, que sigue la Navaja de Ockham: la mejor explicación de unos datos es la que los describe de la forma más simple. Así, la compresión es una aplicación práctica de este principio: el compresor que logra una descripción más corta es también el que mejor entiende los datos.[1]

2.3. Tipos de Compresión

2.3.1. Compresión Sin Pérdida (Lossless)

La compresión sin pérdida se define como el proceso en el que los datos originales pueden ser reconstruidos de forma bit-a-bit idéntica a partir del archivo comprimido. Este tipo de compresión se basa en la eliminación de la redundancia estadística y las repeticiones exactas. Es indispensable para datos donde no se puede tolerar ninguna pérdida de información, como en el caso de archivos de texto, documentos ejecutables, o bases de datos. Ejemplos de formatos que utilizan este tipo de compresión son PNG y ZIP.

2.3.2. Compresión con Pérdida (Lossy)

A diferencia de la compresión sin pérdida, la compresión con pérdida elimina parte de la información para reducir más el tamaño del archivo. Es adecuada para datos en los que una pérdida limitada es aceptable, como en imágenes, audio o vídeo. Este método aprovecha las limitaciones de la percepción humana y la redundancia de la señal. Ejemplos comunes son los formatos JPEG, MP3 y H.264. Su objetivo es mantener un equilibrio entre la tasa de bits y la calidad percibida.

2.4. Métricas de Evaluación

Para evaluar el rendimiento de los algoritmos de compresión, se utilizan varias métricas. La eficiencia se mide por la **Tasa de Compresión**, que es la relación entre el tamaño original y el tamaño comprimido. En el contexto de los benchmarks de compresión de texto, la eficiencia se expresa a menudo en **Bits Per Character (BPC)**, mientras que para imágenes se utiliza **Bits Per Pixel (BPP)**.[1]

Para la compresión con pérdida, donde la reconstrucción no es idéntica al original, se requieren métricas que midan la fidelidad y la calidad percibida. Las más comunes son:

- **Mean Squared Error (MSE):** Mide el error cuadrático medio entre la señal original I y la reconstruida $\hat{I}$. Para una imagen de tamaño $M \times N$, se define como:

$$\text{MSE} = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(I(i,j) - \hat{I}(i,j) \right)^2$$

- **Peak Signal-to-Noise Ratio (PSNR):** Una métrica matemática que mide la relación entre la potencia máxima posible de una señal y la potencia del ruido que la distorsiona. Se expresa en decibelios (dB) y se calcula a partir del MSE:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{\text{MSE}} \right)$$

  donde MAX_I es el valor máximo posible de un píxel (por ejemplo, 255 en imágenes de 8 bits).

3. La Teoría de la Información y Entropía

Claude E. Shannon estableció los fundamentos teóricos de la compresión en 1948 con su publicación *A Mathematical Theory of Communication*.

La Entropía de Shannon, denotada como $H(X)$, mide la incertidumbre de una fuente de información. Una fuente, en este contexto, es un proceso que emite una secuencia de símbolos; en terminología moderna, se modela como un proceso estocástico discreto y finito. La entropía se define como el promedio mínimo de bits necesarios para codificar los símbolos de dicha fuente sin perder información.

La fórmula para una fuente discreta X, con símbolos x_i y probabilidades de ocurrencia $p(x_i)$, es:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{3.1}$$

Para la compresión sin pérdida, la entropía representa el límite teórico. Un compresor ideal alcanzaría una tasa de compresión igual a la entropía de la fuente. Este límite no suele alcanzarse en la práctica, debido a que los modelos de compresión solo aproximan la distribución real de los datos.

4. Métodos Clásicos: Los Pilares de la Compresión

4.1. Algoritmos Sin Pérdida

4.1.1. Codificación de Huffman

La codificación de Huffman es un método de codificación de entropía que asigna códigos de longitud variable a los símbolos de una fuente de datos, de modo que los símbolos más frecuentes tengan códigos más cortos. Esto se logra mediante la construcción de un árbol de prefijos de abajo hacia arriba.
Procedimiento y Pseudocódigo:

1. **Frecuencias:** Se calcula la frecuencia de aparición de cada símbolo en el archivo de entrada.

2. **Construcción del Árbol:** Se crea un nodo hoja para cada símbolo, con su frecuencia como peso. Se insertan estos nodos en una cola de prioridad, de modo que los nodos con menor frecuencia tengan mayor prioridad.

3. **Combinación de Nodos:** Mientras queden más de un nodo en la cola:

   - Se extraen los dos nodos de menor peso.
   - Se crea un nuevo nodo interno, cuyo peso es la suma de los pesos de los dos nodos extraídos. Estos dos nodos se convierten en los hijos del nuevo nodo (el de menor peso, por convención, a la izquierda).
   - El nuevo nodo se inserta en la cola de prioridad, manteniendo el orden.

4. **Generación de Códigos:** Una vez que solo queda un nodo (la raíz del árbol de Huffman), se recorre el árbol desde la raíz hasta cada nodo hoja para asignar los códigos binarios. Los caminos a la izquierda se etiquetan con un '0' y los caminos a la derecha con un '1'. El código de un símbolo es la secuencia de bits acumulada en el camino desde la raíz hasta su nodo hoja.

Algorithm 1 Algoritmo de codificación Huffman

```
function CONSTRUIR_ARBOL_HUFFMAN(frecuencias)
    cola_prioridad ← COLA_DE_PRIORIDAD
    for all simbolo, frecuencia in frecuencias do
        nodo ← NUEVO_NODO(simbolo, frecuencia)
        cola_prioridad.insertar(nodo)
    end for
    while cola_prioridad.tamano() > 1 do
        nodo1 ← cola_prioridad.extraer_min()
        nodo2 ← cola_prioridad.extraer_min()
        nodo_padre ← NUEVO_NODO(NULL, nodo1.frecuencia + nodo2.frecuencia)
        nodo_padre.izquierda ← nodo1
        nodo_padre.derecha ← nodo2
        cola_prioridad.insertar(nodo_padre)
    end while
    return cola_prioridad.extraer_min()
end function
```

4.1.2. Algoritmos de Diccionario: LZ77

Los algoritmos de la familia Lempel-Ziv son métodos de compresión sin pérdida basados en diccionarios. A diferencia de Huffman, no operan sobre la base de la frecuencia de los símbolos individuales, sino que buscan y reemplazan secuencias de bytes repetidas. El algoritmo LZ77 fue uno de los primeros en introducir el concepto de una ventana deslizante.

Procedimiento y Pseudocódigo:

1. **Ventanas:** El algoritmo utiliza dos búferes: una ventana deslizante que contiene los datos ya procesados y una ventana de anticipación que contiene los datos por procesar.

2. **Búsqueda:** El compresor escanea la ventana de anticipación para encontrar la coincidencia de cadena más larga que ya exista en la ventana deslizante.

3. **Codificación:** Si se encuentra una coincidencia:

   - El compresor emite un token de tres partes: (desplazamiento, longitud, literal). El desplazamiento indica la posición de la coincidencia en la ventana deslizante, la longitud es la de la cadena coincidente, y el literal es el primer carácter que no coincide.
   - La ventana deslizante avanza una cantidad de bytes igual a la longitud de la coincidencia más un byte.

4. **Codificación de Literales:** Si no se encuentra ninguna coincidencia:

- El primer carácter de la ventana de anticipación se emite como un literal no comprimido.
- La ventana deslizante avanza un byte.

Este proceso se repite hasta que todo el archivo haya sido procesado.

Algorithm 2 Algoritmo de codificación LZ77

```
function COMPRIMIR_LZ77(entrada, tamano_ventana, tamano_anticipacion)
    salida ← LISTA
    puntero_entrada ← 0
    while puntero_entrada < entrada.tamano() do
        mejor_desplazamiento ← 0
        mejor_longitud ← 0
                                                ▷ Buscar la coincidencia más larga
        puntero_busqueda ← puntero_entrada - tamano_ventana
        if puntero_busqueda < 0 then
            puntero_busqueda ← 0
        end if
        for i from puntero_busqueda to puntero_entrada - 1 do
            longitud_actual ← 0
            while (puntero_entrada + longitud_actual < entrada.tamano()) and (entrada[i + longitud_actual] == entrada[puntero_entrada + longitud_actual]) do
                longitud_actual ← longitud_actual + 1
            end while
            if longitud_actual > mejor_longitud then
                mejor_longitud ← longitud_actual
                mejor_desplazamiento ← puntero_entrada - i
            end if
        end for
                                                ▷ Emitir token
        if mejor_longitud > 0 then
            salida.agregar((mejor_desplazamiento, mejor_longitud, entrada[puntero_entrada + mejor_longitud]))
            puntero_entrada ← puntero_entrada + mejor_longitud + 1
        else
            salida.agregar((0, 0, entrada[puntero_entrada]))
            puntero_entrada ← puntero_entrada + 1
        end if
    end while
    return salida
end function
```

4.1.3. DEFLATE, ZIP y PNG

El algoritmo DEFLATE es una combinación híbrida de los principios de LZ77 y la codificación de Huffman.Fue diseñado por Phil Katz y se especifica en la RFC 1951. La compresión se logra en dos etapas: primero, las cadenas de bytes duplicadas se reemplazan con punteros (desplazamiento, longitud) (la parte LZ77), y luego, estos literales y punteros se codifican utilizando árboles de Huffman. Este enfoque es la base de algunos de los formatos de compresión sin pérdida más comunes, como ZIP, gzip y las imágenes PNG.

4.2. Algoritmos con Pérdida

4.2.1. La Transformada de Coseno Discreta (DCT)

La Transformada de Coseno Discreta (DCT) es una técnica de compresión con pérdida muy utilizada, especialmente en el ámbito de las imágenes y el vídeo. Su principio fundamental es convertir una señal, como los datos de un bloque de píxeles, del dominio espacial al dominio de la frecuencia.
Procedimiento y Pseudocódigo:

1. **División en Bloques:** La imagen de entrada se divide en pequeños bloques de datos, comúnmente de 8×8 o 16×16 píxeles.

2. **Aplicación de la Transformada:** A cada bloque se le aplica la DCT 2D. Esta transformada tiene una propiedad crucial conocida como *compactación de energía*. La mayoría de la información visualmente significativa (detalles de baja frecuencia) se concentra en unos pocos coeficientes en la esquina superior izquierda de la matriz de salida, mientras que los coeficientes de alta frecuencia (detalles finos) tienden a ser cercanos a cero.

La fórmula para la DCT 2D de una matriz de M×N píxeles A_{mn} es:

$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos\left(\frac{\pi(2m+1)p}{2M}\right) \cos\left(\frac{\pi(2n+1)q}{2N}\right) \tag{4.1}$$

Donde α_p y α_q son factores de normalización: $\alpha_p = \sqrt{1/M}$ para $p = 0$ y $\alpha_p = \sqrt{2/M}$ para $p > 0$. La compresión se logra en el siguiente paso.

3. **Cuantización:** Este es el paso con pérdida. Los coeficientes de la matriz B_{pq} se dividen por una matriz de cuantización, que es una matriz de pesos predefinida. Esto reduce la precisión de los coeficientes de alta frecuencia y a menudo los convierte en cero. Estos coeficientes, menos importantes para la percepción humana, se descartan, logrando una reducción masiva de datos. La descompresión utiliza la transformada inversa de la DCT (IDCT) para reconstruir el bloque de píxeles.

4.2.2. Otras Transformadas: Fourier y Ondículas

Aunque la Transformada de Fourier es precursora de la DCT, sus limitaciones con ciertos datos la hacen menos adecuada para la compresión de imágenes. La DCT está

diseñada para superar estos inconvenientes. Una técnica posterior, la Transformada Wavelet, ofrece un mejor rendimiento al analizar la señal en múltiples escalas. Este enfoque captura los detalles y bordes de la imagen con mayor fidelidad y elimina los .ªrtefactos de bloque"que genera la DCT. El formato JPEG 2000, por ejemplo, se basa en Wavelets para conseguir una compresión superior.

4.2.3. Aplicaciones Prácticas: JPEG, MP3, y H.264

Las transformadas matemáticas son fundamentales en los formatos de compresión multimedia más comunes:

- **JPEG:** Este formato de imagen aplica la DCT para compactar la energía visual en pocos coeficientes y luego utiliza la cuantización para eliminar los detalles de alta frecuencia.
- **MP3 / AAC:** Estos formatos de audio usan la MDCT para descomponer la señal en frecuencias. Posteriormente, los modelos psicoacústicos identifican y descartan los componentes que el sistema auditivo humano no puede percibir (enmascaramiento).
- **H.264 / MPEG:** Los estándares de vídeo como estos aplican la DCT para la compresión espacial de fotogramas clave o I-frames.

5. Compresión con Inteligencia Artificial y Estado del Arte

5.1. Autoencoders y Variational Autoencoders (VAEs)

Un autoencoder es una red neuronal no supervisada diseñada para aprender una representación compacta y comprimida de datos, conocida como representación latente. Su arquitectura se divide en dos componentes principales: el **codificador**, que mapea la entrada a un espacio de menor dimensión, y el **decodificador**, que reconstruye la entrada a partir de la representación latente. El objetivo del autoencoder es minimizar el error de reconstrucción, forzando al modelo a capturar la información más relevante de los datos en un formato comprimido.

Los autoencoders variacionales (VAEs) extienden este concepto al introducir un enfoque probabilístico. En lugar de aprender un punto fijo en el espacio latente, el codificador de un VAE aprende los parámetros (media y log-varianza) de una distribución de probabilidad (normal multivariada) para cada punto de entrada. El decodificador luego muestrea esta distribución para generar la reconstrucción. Esta característica permite a los VAEs no solo comprimir y reconstruir, sino también generar nuevas imágenes plausibles al muestrear del espacio latente aprendido. Para lograr esto, el entrenamiento de un VAE minimiza una función de pérdida que combina el error de reconstrucción con la divergencia de Kullback-Leibler, una medida de cuán diferente es la distribución latente aprendida de una distribución normal estándar.

5.2. Overfitting como Mecanismo de Compresión

Tradicionalmente, el overfitting es un fenómeno indeseable en el que un modelo aprende el ruido y las particularidades del conjunto de datos de entrenamiento en lugar de capturar los patrones subyacentes y generalizables. Sin embargo, en el contexto de este proyecto, el overfitting se reevalúa como un método intencional de çompresión con pérdida". Si un conjunto de datos es lo suficientemente pequeño, una red neuronal con una capacidad limitada puede memorizar cada una de las entradas. En este escenario, el conjunto de pesos del modelo se convierte, de hecho, en un archivo comprimido de los datos originales, ya que el modelo solo puede generar esas imágenes específicas y no otras. La fidelidad de la reconstrucción es una medida del éxito de esta memorización, y el tamaño del modelo en disco, una medida de la compresión lograda.

5.3. Técnicas de Fine-Tuning Eficiente de Parámetros (PEFT)

El fine-tuning de modelos masivos, como los grandes modelos de lenguaje (LLMs) o los modelos de visión, es un proceso costoso y que requiere una enorme cantidad de recursos computacionales. Para abordar este problema, han surgido las técnicas de **Parameter-Efficient Fine-Tuning (PEFT)**. Estos métodos buscan adaptar un modelo pre-entrenado a una tarea específica modificando solo una pequeña fracción de sus parámetros, en lugar de recalcular todos los pesos.

Una de las técnicas más prominentes es la **Low-Rank Adaptation (LoRA)**. Su funcionamiento se basa en la observación de que los cambios en los pesos de un modelo durante el fine-tuning tienen inherentemente una estructura de bajo rango. LoRA congela los pesos originales del modelo pre-entrenado, que se denotan como W, y en cada capa de atención o lineal, introduce un par de matrices auxiliares, A y B. Estas matrices son de dimensiones significativamente más pequeñas que la matriz original. El ajuste de peso ΔW se calcula como el producto de estas dos matrices:

$$W' = W + \Delta W \quad \text{donde:} \quad \Delta W = BA \tag{5.1}$$

Aquí, W es la matriz de pesos original de tamaño $d \times k$, y las matrices de adaptación son B de tamaño $d \times r$ y A de tamaño $r \times k$. El valor r (el rango) es una hiperparámetro que se mantiene muy pequeño, típicamente entre 4 y 32. Durante el entrenamiento, solo se actualizan los parámetros de las matrices A y B, manteniendo W congelada. El número de parámetros a entrenar es drásticamente reducido, por ejemplo, en el caso de GPT-3, la cantidad de parámetros entrenables se reduce de 175 mil millones a aproximadamente 18 millones, lo que también reduce los requisitos de memoria de GPU en dos tercios.

La eficacia de LoRA se basa en explotar la estructura de bajo rango de los modelos grandes, lo que permite que una pequeña modificación de parámetros genere un cambio funcional significativo para una tarea específica. Tras el entrenamiento, los adaptadores LoRA se fusionan con el modelo base sin añadir latencia durante la inferencia.

6. Metodología

6.1. Descripción Técnica

Para los experimentos iniciales se utilizaron autoencoders con arquitecturas convolucionales. El autoencoder simple se construyó con un codificador que consistía en capas convolucionales seguidas de capas de agrupación (pooling) para reducir la dimensionalidad de las imágenes. El decodificador utilizó capas de deconvolución (transposed convolutions) para reconstruir la imagen a partir del espacio latente. La función de pérdida utilizada fue el Error Cuadrático Medio (MSE) y el Proporción Máxima de Señal a Ruido (PSNR), mencionadas previamente,

El VAE, por su parte, mantuvo una estructura similar en sus capas convolucionales, pero su codificador fue modificado para producir dos vectores para cada entrada: uno que representa la media (μ) y otro para la log-varianza ($\log \sigma^2$) de la distribución latente. Se utilizó la técnica de reparametrización para muestrear del espacio latente y permitir el flujo de gradientes a través de la red durante el entrenamiento. La función de pérdida combinó el MSE para la reconstrucción con la divergencia de Kullback-Leibler (KL) para regular el espacio latente.

6.2. Diseño del Modelo

El `yosemite_model` fue diseñado a partir de un proceso de optimización iterativa. El objetivo era crear el modelo más ligero posible que aún fuera capaz de memorizar las 44 imágenes del conjunto de datos de Yosemite. Las técnicas de reducción de parámetros incluyeron: la disminución del número de filtros en las capas convolucionales, la reducción de la profundidad del modelo y la minimización del tamaño de las capas densas en la parte del codificador. El modelo final fue una arquitectura que, a diferencia de los modelos de autoencoder generalistas, tenía como única función la reconstrucción perfecta de las imágenes de su dataset de entrenamiento.

6.3. Entrenamiento y Validación

Se utilizó el optimizador Adam. La estrategia de entrenamiento del `yosemite_model` fue entrenarse hasta alcanzar un punto de sobreajuste total, donde el error de reconstrucción en el conjunto de entrenamiento tratara de ser nulo. Esta estrategia buscó que el modelo memorizara los datos, con el objetivo de que sus pesos representaran la çompresión"de las imágenes. Por lo tanto, no hubo ningun proceso de validación ya que no se buscaba extrapolar a otros datasets.

7. Experimentos

7.1. Objetivos

Los experimentos se diseñaron para cuantificar el rendimiento de cada modelo y comparar su eficiencia en términos de tamaño, parámetros y error de reconstrucción, lo que permite evaluar la viabilidad de la compresión por overfitting.

7.2. Conjunto de Datos

Se emplearon dos conjuntos de datos: inicialmente se entrenó con MNIST, un conjunto de datos estándar de dígitos escritos a mano, para la validación de los autoencoders genéricos, y posteriormente un conjunto de datos personalizado de 44 imágenes de paisajes de Yosemite.

Cálculo Detallado del Tamaño del Dataset de Yosemite (sin comprimir):

- Número de imágenes: 44
- Resolución: 512 píxeles de alto x 512 píxeles de ancho
- Canales de color: 3 (RGB)
- Profundidad de color: 8 bits por canal (1 byte por píxel por canal)

El cálculo del tamaño total en bytes es:

$$44 \times 512 \times 512 \times 3 \text{ bytes} = 34{,}502{,}592 \text{ bytes}$$

Este valor es equivalente a aproximadamente 34.5 megabytes (MB).

7.3. Configuración

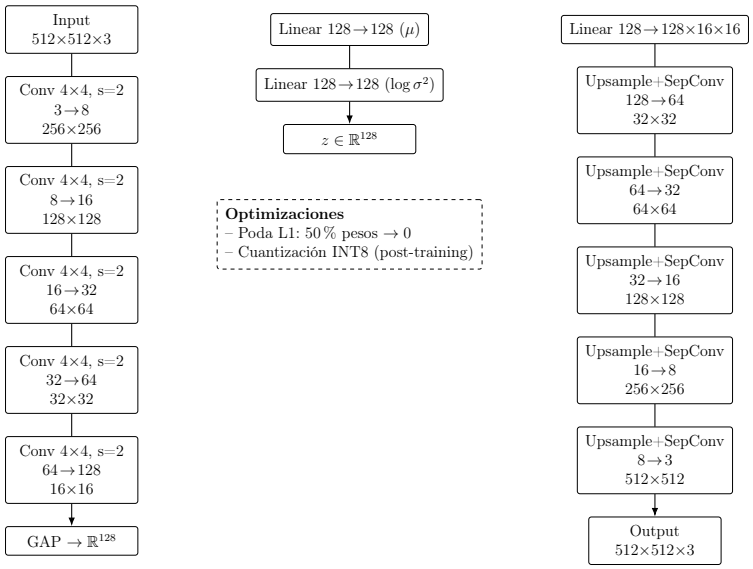


Figura 4: Esquema de la arquitectura del modelo autoencoder

Se investigaron distintas técnicas para reducir el número de parámetros y el tamaño del modelo: la reducción de canales en cada capa, la eliminación del *bottleneck* denso mediante global average pooling o convoluciones 1×1, el uso de convoluciones depthwise/separables, un mayor downsampling intermedio, la cuantización en formatos INT8 o FP16, la poda de pesos, la factorización de matrices y la compartición de parámetros, el uso de tablas de latentes preaprendidos y la reducción de la precisión de activaciones.

7.4. Análisis de Rendimiento

El mayor cuello de botella del modelo es el tiempo de entrenamiento, cuanto más optimizado estaba el modelo en cuanto a tamaño, más tardaba en llegar al overfitting. Sin embargo, el objetivo del proyecto era el ratio de compresión, no la velocidad del algoritmo

7.5. Visualización

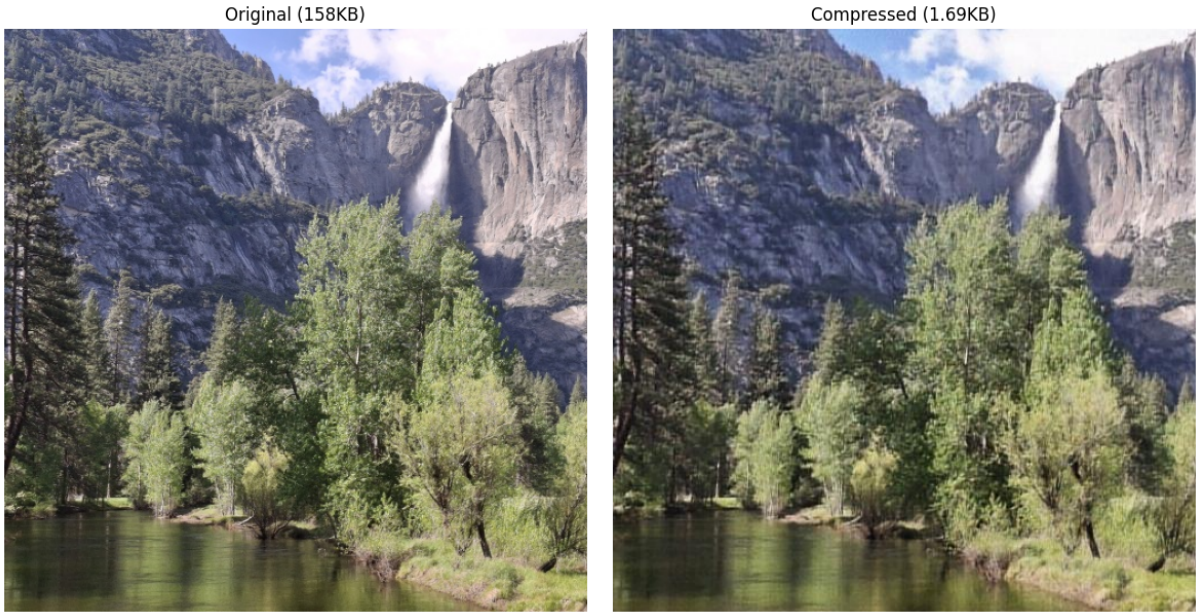


Figura 5: Ejemplo ilustrativo del resultado: reconstrucciones de alta fidelidad.

8. Resultados

Modelo	Entrenamiento	MSE	PSNR (dB)	Parámetros	Tamaño en disco	Compresión
VAE original	10 min	512.7	21.0	18.2 M	69.6 MB	0.50
VAE reducido (canales menores)	19 min	893.4	18.6	4.5 M	17.0 MB	2.03
VAE optimizado (sep. conv + GAP)	17 min	742.9	19.4	4.4 M	16.9 MB	2.04
VAE optimizado + cuantización	18 min	689.2	19.8	4.4 M	4.2 MB	8.21
VAE optimizado + poda + cuantización	20 min	523.6	20.9	4.4 M (50 % ceros)	2.1 MB (sparse)	16.43

Tabla 3: Comparación entre el VAE original y las variantes optimizadas

De todas las optimizaciones, las más eficaces fueron las convoluciones separables, el global average pooling, la cuantización y la poda. El dataset utilizado fue de 44 imágenes RGB de 512×512 píxeles (33 MB). El modelo final (4 MB) resultó más pequeño que el propio conjunto de datos, cumpliendo el objetivo de compresión.

9. Conclusiones y Trabajos Futuros

Conclusiones

Se ha demostrado que un **autoencoder compacto** puede actuar como un compresor específico para un conjunto reducido de imágenes, ocupando menos espacio que el dataset original al que se sobreentrena. Esta aproximación está fundamentada en el teorema de aproximación universal y conecta directamente con los límites teóricos de la compresión.

Aunque se exploraron múltiples optimizaciones, las más efectivas fueron las **convoluciones separables**, el **global average pooling**, la **cuantización** y la **poda**.

Trabajos Futuros

Una línea prometedora consiste en entrenar un **modelo grande** sobre un conjunto variado de imágenes para aprender representaciones generales, y posteriormente aplicar **fine-tuning ligero** con técnicas como LoRA en datasets pequeños. Con ello se almacenaría un único modelo base y cada nuevo conjunto de imágenes podría comprimirse añadiendo muy pocos parámetros extra y con bajo coste computacional, resultando más eficiente a largo plazo.

Referencias

[1] M. Mahoney, *Data Compression Explained*, 2010. Disponible en: https://mattmahoney.net/dc/dce.html

[2] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, N. Johnston, *Variational Image Compression with a Scale Hyperprior*, 2018. Disponible en: https://arxiv.org/pdf/1802.01436v2

[3] A. X. et al., *LLMZip: Lossless Text Compression using Large Language Models*, 2023. Disponible en: https://arxiv.org/pdf/2306.04050v2

[4] T. van Rozendaal, I. A. M. Huijben, T. S. Cohen, *Overfitting for Fun and Profit: Instance-Adaptive Data Compression*, 2021. Disponible en: https://arxiv.org/pdf/2101.08687

[5] M. Hutter, *500,000€ Prize for Compressing Human Knowledge*, 2006. Disponible en: http://prize.hutter1.net/index.htm

[6] F. Bellard, *NNCP v2: Lossless Data Compression with Transformer*, 2023. Disponible en: https://bellard.org/nncp/nncp_v2.pdf