ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENIERIA EN SISTEMAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

COMPRESIÓN DE DATOS CLIMÁTICOS MEDIANTE TÉCNICAS DE DEEP LEARNING

Autor: Diego Benítez Elías

Tutor: Leopoldo Carro Calvo

Curso académico: 2023 /2024

TÍTULO: *Compresión de datos climáticos mediante técnicas de Deep Learning*

AUTOR: *Diego Benítez Elías*

TUTOR: *Leopoldo Carro Calvo*

CO-TUTOR:

La defensa del presente Trabajo Fin de Grado se realizó el día

siendo calificada por el siguiente tribunal:

PRESIDENTE:

SECRETARIO:

VOCAL:

Habiendo obtenido la siguiente calificación:

CALIFICACIÓN:

Presidente **Secretario** **Vocal**

Resumen

Los datos climáticos utilizados por los científicos para el análisis del clima pasado, presente y futuro, representan una gran cantidad en términos de recursos de almacenamiento y cómputo. Es por ello que se hace imprescindible un tratamiento de los mismos que mitigue estos efectos. Para ello, se ha propuesto la compresión con pérdidas de los datos asegurando una reconstrucción al menos estadísticamente indistinguible de la variabilidad natural del sistema para posteriores análisis y visualización por parte de la comunidad científica. En este proyecto se han evaluado técnicas de *Deep Learning* (aprendizaje profundo) para el desarrollo de un modelo de compresión con pérdidas que permita alcanzar los objetivos y requerimientos de la reconstrucción de los datos climáticos. Se ha desarrollado un modelo de autoencoder convolucional que, junto con un tratamiento de la compresión de la predicción de los datos de entrada y los residuos de ésta, cumple con los anteriores requerimientos.

The climate data used by scientists for the analysis of past, present, and future climates represent a significant amount in terms of storage and computing resources. Therefore, it is essential to manage this data to mitigate these effects. Lossy compression has been proposed to ensure a reconstruction that is at least statistically indistinguishable from the natural variability of the climate system for subsequent analysis and visualization by the scientific community. In this project, Deep Learning techniques have been evaluated to develop a lossy compression model that meets the objectives and requirements for reconstructing climate data. A convolutional autoencoder model has been proposed, which, along with the treatment of input prediction compression data and its residuals, meets these requirements.

Agradecimientos

A mi madre y en memoria de mi padre. A mi familia, amigos y compañeros por su compañía, ayuda y comprensión. Al profesor Leopoldo Carro, por su inestimable ayuda, guía, conocimiento, paciencia y atención.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Descripción del problema

Las simulaciones climáticas de alta resolución, utilizadas para el estudio del clima pasado, presente y futuro, requieren grandes cantidades de recursos en cómputo y generan ingentes cantidades de datos. El Community Earth System Model (CESM) es un popular y ampliamente utilizado modelo de la tierra cuyo desarrollo está tutelado por el National Center for Atmospheric Research (NCAR). Las simulaciones del CESM pueden fácilmente superar el terabyte de datos de cómputo por día. De hecho, se espera que las simulaciones del Coupled Model Comparison Project (Phase 6) del CESM excedan los 10 petabytes de datos en bruto. [1]

Los datos generados por estas simulaciones se almacenan para análisis posteriores y son a menudo insostenibles en términos de recursos, intercediendo negativamente en las metas científicas al tener que reducir, por ejemplo, los tamaños de la simulación o la frecuencia de las muestras de los datos obtenidos. El histórico de los datos del proyecto CESM, o datos en bruto en formato NetCDF [2], son transformados desde una precisión de 64 bits a una de 32 bits. Los científicos climáticos se muestran reacios a aplicar una compresión adicional de estos datos, sin embargo, no parece haber razones de peso para afirmar que la precisión de 32 bits sea la más óptima para una variable particular [3]. Es por ello que podría ser interesante una compresión que redujese los bits menos significativos y que permitiese una posterior reconstrucción, al menos estadísticamente indistinguible, en el análisis post-proceso, tanto en términos analíticos y de visualización, de la variabilidad natural del sistema climático [4].

1.2. Objetivos del trabajo

El objetivo de este trabajo fue analizar el uso de modelos de aprendizaje profundo (*deep learning*)sobre conjuntos de datos climáticos como técnica de compresión con pérdidas como alternativa a los métodos de compresión utilizados hasta la fecha, haciendo uso de las métricas de calidad utilizadas en [4] para asegurar una apropiada reconstrucción de los datos originales.

1.3. Estructura del documento

En el presente documento se ha abordado el estudio de la problemática sobre el impacto generado, en términos de recursos, por los datos climáticos utilizados por los científicos, así como se detalla el desarrollo de un modelo de compresión con pérdidas utilizando técnicas de aprendizaje profundo. En el **apartado 1** se describe el problema que supone el tratamiento de estos datos en cuanto al almacenamiento y cómputo de los mismos, además de manifestar el objetivo del proyecto. En el **apartado 2** se presenta un estudio del estado del arte de la literatura acerca de los diferentes métodos de compresión en el contexto de las variables climáticas y se exponen algunas técnicas de aprendizaje profundo utilizadas.

El **apartado 3** ofrece una descripción, a modo de introducción, de las características y elementos más importantes del aprendizaje profundo, finalizando con una visión de los autoencoders, modelo utilizado en el análisis del proyecto. La metodología llevada a cabo para el desarrollo del trabajo se describe en el **apartado 4**. El modelo, detalles y arquitectura del autoencoder experimento sobre el que se probará la solución y se evaluarán los resultados, es descrito en el **apartado 5**, junto con la descripción de los procesos de compresión y reconstrucción del modelo de compresión desarrollado en el proyecto, detallando el tratamiento de los datos comprimidos y los residuos. En el **apartado 6** se especifican las características de los conjuntos de datos climáticos utilizados durante el entrenamiento y prueba del modelo, el preprocesado de los mismos y los criterios de evaluación que se tendrán para el análisis posterior de los resultados. También en este apartado se detalla el entorno de ejecución del experimento y las diferentes técnicas que se usaron para la obtención de la solución.

Finalmente, en el **apartado 7** se presentan los resultados obtenidos como consecuencia de la ejecución del experimento y, en el **apartado 8**, se tratan las conclusiones desprendidas de la elaboración del proyecto, junto con algunas posibles líneas futuras.

Capítulo 2

Estado del Arte

2.1. Revisión de literatura sobre compresión de datos climáticos

La compresión de datos es un problema fundamental y bien estudiado en ingeniería, siendo comúnmente formulado con el objetivo de diseñar codificaciones para un conjunto discreto de los datos con una mínima entropía [5]. Los compresores sin pérdidas de propósito general proporcionan una escasa reducción de los datos representados en coma flotante, ya que la compresión aprovecha patrones repetitivos y éstos presentan una menor redundancia debido a una naturaleza inherentemente aleatoria. Es por todo ello más interesante el uso de una compresión con pérdidas, donde los datos no se recuperan exactamente sino que se aproximan, lo cual puede ser aceptable debido a que los propios datos están ya sujetos a errores en la medida o en la precisión. [6]

En los trabajos de Baker et. al se muestran una serie de métricas que permiten evaluar el impacto que tienen diferentes técnicas de compresión con pérdidas en la integridad de la reconstrucción de los datos climáticos [4] , así como una evaluación por parte de los científicos de los efectos de la compresión [3]. La naturaleza de los datos climáticos y su diversidad hacen que sea crucial la elección de un método de compresión concreto o unas métricas adecuadas para el posterior análisis de características relevantes [7]. Se puede determinar que, en general, la compresión con pérdidas de los datos climáticos presenta ventajas en la reducción del volumen de éstos sin impactar negativamente en las conclusiones científicas [3].

En Baker -2014, debido a la naturaleza diversa de las diferentes variables a analizar, y con el objetivo de evaluar las métricas anteriormente mencionadas, se seleccionan los siguientes algoritmos de compresión:

Method	lossless mode	special values	freely avail.	fixed quality	fixed CR	32- & 64-bit
GRIB2 + jpeg2000	N	Y	Y	N	N	N
APAX	Y[1]	N	N	Y	Y	Y
fpzip	Y	N	Y	N	N	Y
ISABELA	N	N	Y	N	N	Y

Tabla 2.1: Tabla con las propiedades de los diferentes algoritmos analizados en [4]

En la tabla 2.1 se muestra un resumen de algunas de las características de los algoritmos escogidos para la evaluación. Se utilizaron dos niveles de precisión diferentes para el algoritmo FPZIP (16 y 24 bits), tres configuraciones de ratio de compresión para ISABELLA (2,4 y 5) y tres configuraciones diferentes de error relativo para APAX (1.0, 0.5 y 0.1).

	GRIB2	*ISABELA*	*fpzip*	*APAX*	NC
avg. CR	0.37	0.42	0.18	0.29	0.61
best CR	0.03	0.20	0.02	0.06	0.07
worst CR	0.86	0.77	0.68	0.80	0.86
avg. ρ	.9999999	.9999991	.9999995	.9999991	1.0
avg. nrmse	5.73e-5	3.22e-4	2.35e-4	2.61e-4	0.0
avg. e_{nmax}	1.01e-4	5.56e-3	2.76e-3	1.83e-3	0.0

Tabla 2.2: Tabla con los resultados de compresión de los algoritmos analizados en [4]

Desarrollaron un método híbrido al combinar las distintas configuraciones de los algoritmos con las variables más adecuadas para maximizar los resultados en términos de ratio de compresión y coefiente de correlación. Como puede verse en la tabla 2.2, los mejores resultados se obtuvieron del algoritmo fpzip seguido de APAX.

En un trabajo posterior, Baker-2016 [3], también evaluán las pérdidas de compresión en datos climáticos usando fzip como algoritmo de compresión. La tabla 2.3 muestra los diferentes ratios de compresión en diferentes frecuencias de obtención de los datos haciendo uso de la compresión con pérdidas fpzip, la compresión sin pérdidas NetCDF-4 y el truncamiento de los datos al mismo nivel de precisión que el especificado para fpzip. Podemos ver la ventaja en términos de ratio de compresión que ofrece fpzip.

Method	Monthly	Daily	6-hourly	Average
fpzip	.15	.22	.18	.18
NetCDF-4	.51	.70	.63	.62
Truncation	.61	.58	.60	.69

Tabla 2.3: Tabla con los ratios de compresión en diferentes frecuencias de recogida de los algoritmos analizados en [4]

En el trabajo [7] de Baker et al., se evaluaron los compresores fpzip y SPECK, con distintas configuraciones, sobre diferentes variables climáticas del modelo CESM, mostrando SPECK un mejor desempleño en términos de ratio de compresión que fpzip, como se muestra en la tabla 2.4.

variable name	*SPECK*				*fpzip*				DWT→IDWT
	variant	e_{nmax}	*nrmse*	*CR*	variant	e_{nmax}	*nrmse*	*CR*	max. abs. error
H2O2	speck_2	2.47e-4	2.47e-5	.06	fpzip_20	2.56e-4	2.05e-5	.23	0.0
FSNTC	speck_8	1.75e-4	2.08e-5	.26	fpzip_24	2.33e-5	1.18e-5	.36	0.0
TS	speck_4	1.46e-3	1.95e-4	.13	fpzip_24	8.71e-5	4.95e-5	.28	0.0
TAUY	speck_12	8.04e-6	1.24e-6	.38	fpzip_24	1.41e-5	7.92e-7	.54	0.0
CLOUD	–	–	–	–	fpzip_24	1.70e-5	2.42e-6	.36	8.88e-16
PRECSC	–	–	–	–	fpzip_16	4.01e-3	1.97e-4	.12	2.53e-24
TOT_ICLD_VISTAU	–	–	–	–	fpzip_24	2.68e-5	5.84e-7	.38	8.88e-15
PRECCDZM	speck_24	7.44e-9	1.61e-9	.77	fpzip_16	3.89e-3	4.16e-4	.24	5.29e-23
OMEGAT	speck_16	2.24e-8	3.11e-9	.51	fpzip_24	1.09e-5	2.04e-7	.52	0.0
FLNS	speck_12	1.38e-5	2.72e-6	.38	fpzip_24	2.81e-5	5.19e-6	.42	0.0
VQ	speck_16	3.50e-9	3.82e-10	.51	fpzip_24	9.53e-6	6.52e-7	.48	0.0
NUMLIQ	–	–	–	–	fpzip_32	0.0	0.0	.46	5.96e-8
WSUB	–	–	–	–	fpzip_32	0.0	0.0	.43	0.0

Tabla 2.4: Tabla comparativa de los ratios de compresión CR entre los compresores SPECK y fpzip [7]

Fpzip es un compresor predictivo para campos escalares, reduce los datos debido a una codificación de entropía de los residuos entre la predición y los valores actuales. Fue diseñado para una compresión sin pérdidas. Su extensión con pérdidas se basa en el truncamiento de los valores en coma flotante haciendo cero un número de bits menos significativos de la mantisa. Los bits restantes truncados son comprimidos sin pérdidas [6] [8].

2.2. Aplicación de métodos de aprendizaje automático a la compresión de datos climáticos e imágenes

La problemática de la reducción de dimensionalidad de los datos climáticos generados a partir de modelos más complejos mediante técnicas de aprendizaje automático y, para el interés de este proyecto, mediante técnicas de aprendizaje profundo, ha sido abordada anteriormente en trabajos como [9], donde utilizan autoencoders convolucionales para comprimir datos de la temperatura de la superficie procedente de los datasets CCSM4-T31 y IPSL-CM5ALR. En la figura 2.1 podemos ver un ejemplo de una muestra original y dos reconstrucciones procedentes de diferentes modelos.

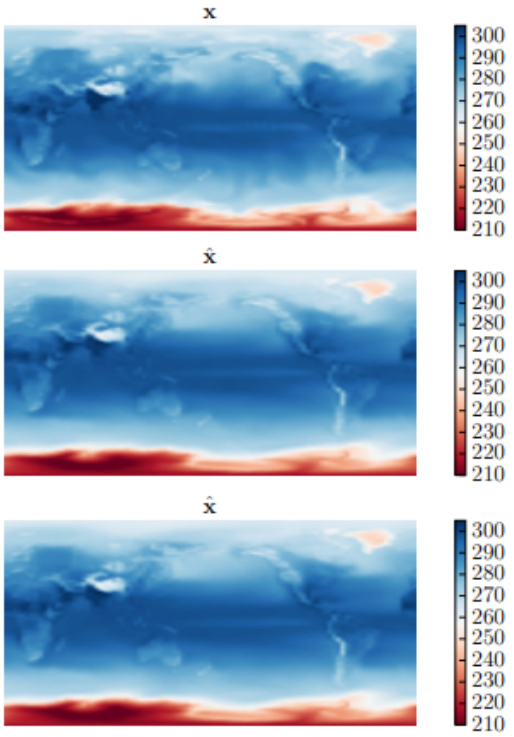

Figura 2.1: Comparación de una muestra original de temperatura y dos reconstrucciones procedentes de diferentes modelos del dataset IPSL-CM5A-LR [9]

En [10] se propone un modelo climático reducido, a partir de simulaciones de datos del dataset ERA5 reanalysis, construido usando redes neuronales convolucionales. La arquitectura de red que utilizan está basada en las redes ResNet. Otro estudio que utiliza redes convolucionales para datos de temperatura es [11], el cual detalla el proceso de muestreo de los datos climáticos de la temperatura de la superficie (2m) a partir del dataset Coupled Model Intercomparison Project Phase 5 (CMIP5). En el citado trabajo encuentran que los autoencoders sobresalen en la generación de proyecciones climáticas especialmente para la temperatura de la superficie. Por último, en [12] proponen el uso de un autoencoder convolucional para aprender una reprensentación compacta de entradas con información espacial para el post-procesado de modelos; en este caso, un modelo de la predicción de la temperatura en la superficie (2m) de estaciones en Alemania.

El estudio [13] muestra el uso de un autoencoder convolucional para la compresión de una señal de electrocardiograma (ECG). Dicho trabajo utiliza un autoencoder con 27 capas que reduce los datos de entrada para su posterior reconstrucción.

En cuanto a técnicas de aprendizaje profundo en la compresión de imágenes, destaca el uso de modelos generativos para la compresión como el expuesto en [14], aprovechando la adaptabilidad a la estructura de los datos que esta técnica permite y mediante una red DCGAN (Deep Convolutional Generative Adversarial Network) consiguiendo mayor compresión que JPEG2000. En [5] proponen un método de compresión basado en una codificación no lineal con el objetivo de mejorar el ratio-distorsión mediante el uso de

un autoencoder generativo variacional (VAE). En el trabajo [15] elaboran un modelo de autoencoder convolucional para la compresión con pérdidas de imágenes, afirmando que este enfoque se comporta mejor que los basados en redes recurrentes.

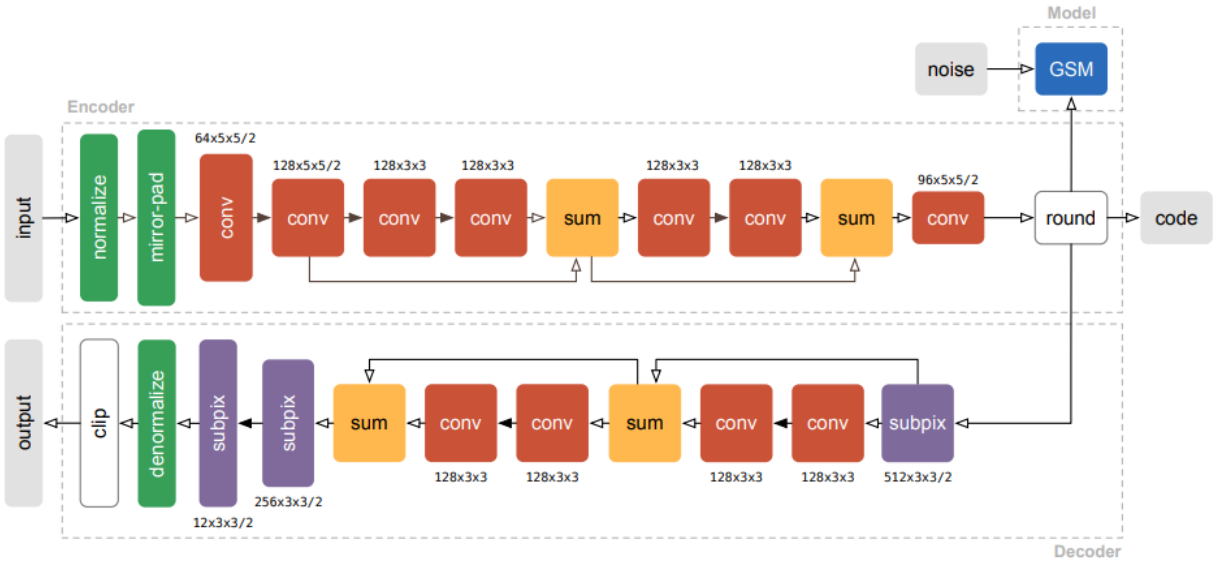

Figura 2.2: Modelo de arquitectura del autoencoder utilizado en [15]

Capítulo 3

Autoencoders como métodos de compresión de la información

En el presente trabajo se ha estudiado el uso de los autoencoders como método de compresión. Los autoencoders pertenecen al campo del aprendizaje profundo o *deep learning*, el cual es a su vez un conjunto de algoritmos del aprendizaje automático o *machine learning*. Ambos pertenecen a la rama de la inteligencia artificial. El uso del aprendizaje profundo y sus diferentes modelos ha ido ganando popularidad y utilidad debido, en parte, a algunos aspectos clave como una mayor digitalización de la sociedad, un incremento de los datos y bases de datos para un mejor entrenamiento, una mejora del *hardware* y el *software* y una mejora en la precisión de las soluciones para aplicaciones cada vez más complejas [16].

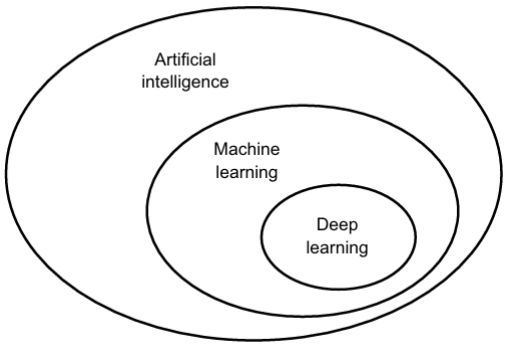

Figura 3.1: Diagrama de los campos de la inteligencia artificial, aprendizaje automático y el aprendizaje profundo [17]

El enfoque central del aprendizaje automático y el aprendizaje profundo es la transformación significativa de los datos, es decir, aprender representaciones de los datos de entrada para una mejor aproximación a los datos de salida [17]. Aunque no es el objeto de este trabajo presentar una descripción detallada de todas las técnicas utilizadas para

la realización del experimento, podría ser conveniente introducir una noción general de algunas de ellas antes de formular la solución propuesta.

3.1. Redes neuronales

Las redes neuronales son un modelo de aprendizaje automático inspirado en las redes neuronales biológicas [18] pero no son modelos del cerebro humano, sino herramientas matemáticas que nos permiten construir representaciones de los datos. Son una herramienta con capacidad para aprender, generalizar y procesar información [19]. La forma más simple de red neuronal es la formada por una neurona, la cual es una unidad computacional que recibe la información por unas entradas y computa su salida tras realizar una operación de regresión junto a una posterior función de transferencia (función de activación) [19] [20], esto permite tratar la no linealidad de las representaciones.

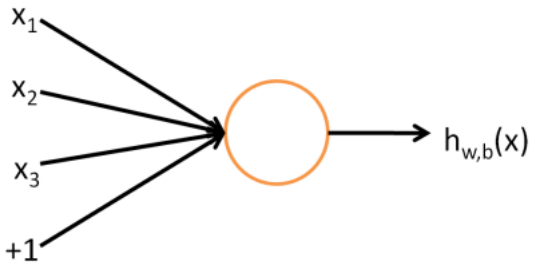

Figura 3.2: Ilustración de una neurona [20]

La formulación matemática de la neurona simple puede verse como:

$$h_{W,b}(x) = f\left(W^T x\right) = f\left(\sum_{i=1}^{3} W_i^T x_i + b\right)$$

donde $f : \mathbb{R} \mapsto \mathbb{R}$ es la función de activación, W son los pesos o parámetros ajustables y b es el sesgo. Algunos ejemplos de funciones de activación ampliamente usadas se pueden ver en la figura 3.3.

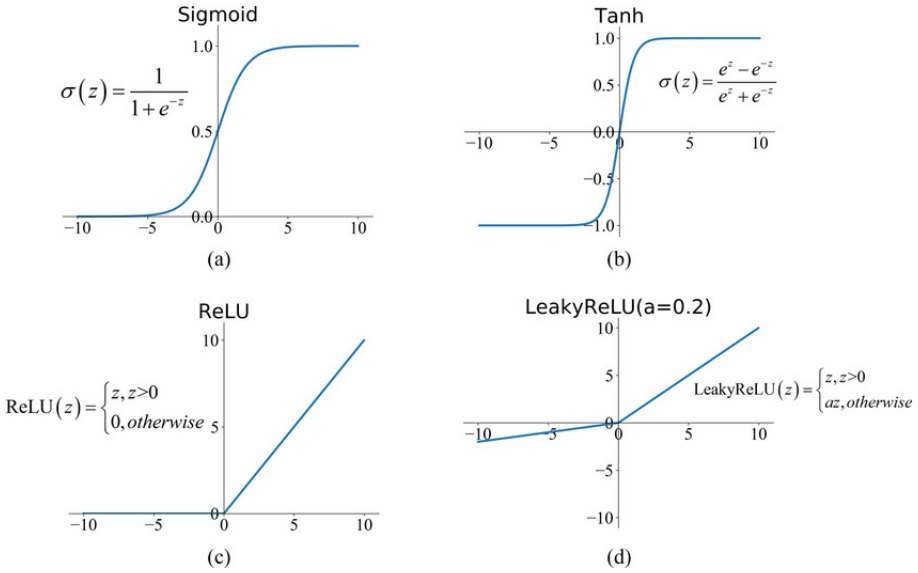

Figura 3.3: Algunas funciones de activación (a) Sigmoid, (b) Tanh, (c) ReLU, y (d) Leaky-ReLU [21]

Si agrupamos e interconectamos algunas de estas neuronas simples, podemos conseguir una red neuronal simple:

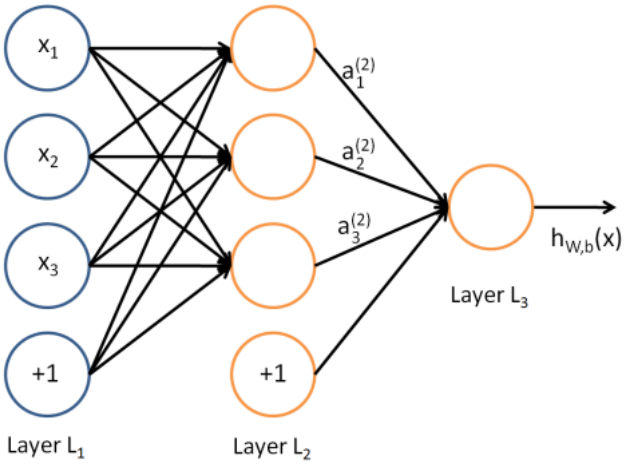

Figura 3.4: Red neuronal simple [20]

caracterizada por:

$$a_1^{(2)} = \mathrm{f}(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}$$

$$a_2^{(2)} = \mathrm{f}(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = \mathrm{f}(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_2^{(1)})$$

$$h_{W,b}(x) = a_1^{(3)} = f\left(W_{11}^{(2)}a_1^{(2)} + W_{11}^{(2)}a_2^{(2)}W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}\right)$$

3.2. Redes convolucionales

Las redes convolucionales o *convolutional neural networks* (CNN) son un tipo de redes neuronales que usan la operación convolución en al menos una de sus capas en el lugar de la matriz de multiplicación [16]. Están orientadas a procesar datos con una estructura de cuadrícula, como las imágenes (2D) o las series temporales (1D). Las capas convolucionales aprenden y forman representaciones de los datos de entrada aplicando una convolución con filtros sobre los datos de entrada, generando mapas de características. La matriz de convolución o *kernel*, en este caso es una matriz de pesos que permite computar información y proveer características tales como bordes o curvas [22]. La salida de una capa convolucional se verá afectada tanto por la forma de su entrada, como por la elección de la forma del *kernel,* su *padding* o sus *strides [23].*

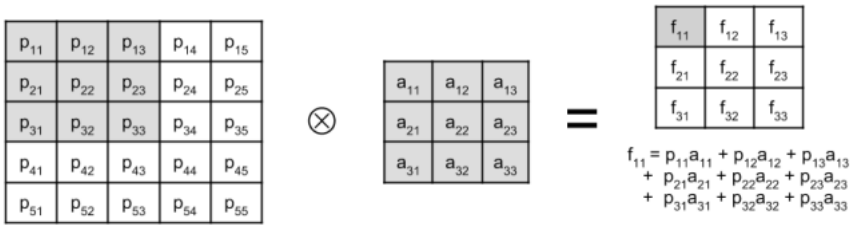

Figura 3.5: Computación de la operación convolución [24]

La convolución es también más eficiente que una matriz de multiplicación densa en términos de requerimiento de memoria y eficiencia estadística [16]. Una diferencia fundamental respecto a las capas totalmente conectadas (*Dense*) es que éstas aprenden patrones globales mientras que las capas convolucionales aprenden también patrones locales [17]. La figura 3.6 muestra la estructura tradicional de una red convolucional. Las capas convolucionales están constituidas por varias unidades convolucionales cuyos parámetros son optimizados mediante el algoritmo de retropropagación [25].

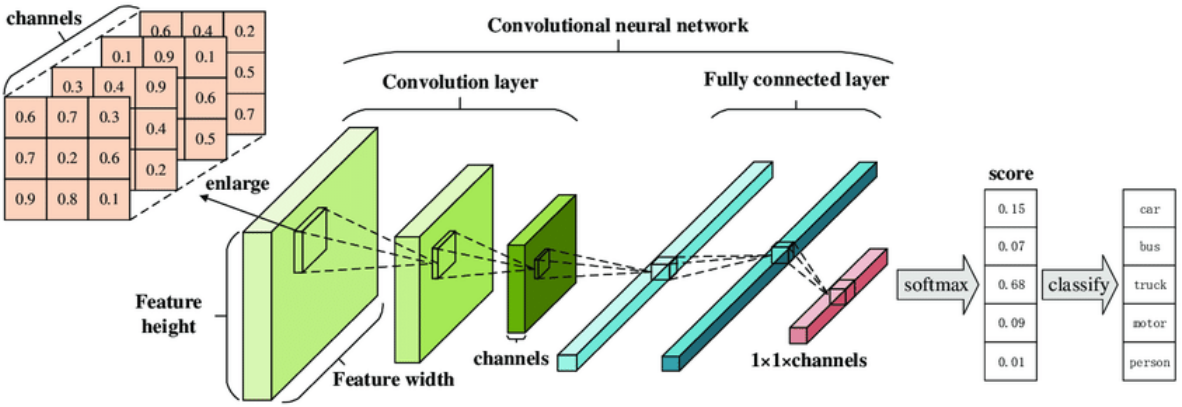

Figura 3.6: Estructura de una red convolucional para clasificación [25]

Otras características importantes de las redes convolucionales son [17]:

- *Los patrones que aprenden son invariantes a la traslacion*, es decir, cualquier patrón que las capas convolucionales aprendan, puedan reconocerlo en cualquier otro sitio en una nueva localización.
- *Pueden aprender patrones espaciales jerárquicamente*, o dicho de otro modo, las primeras capas convolucionales pueden aprender patrones locales pequeños como bordes y después usar esta información para, en capas posteriores, aprender otros patrones derivados de éstos.

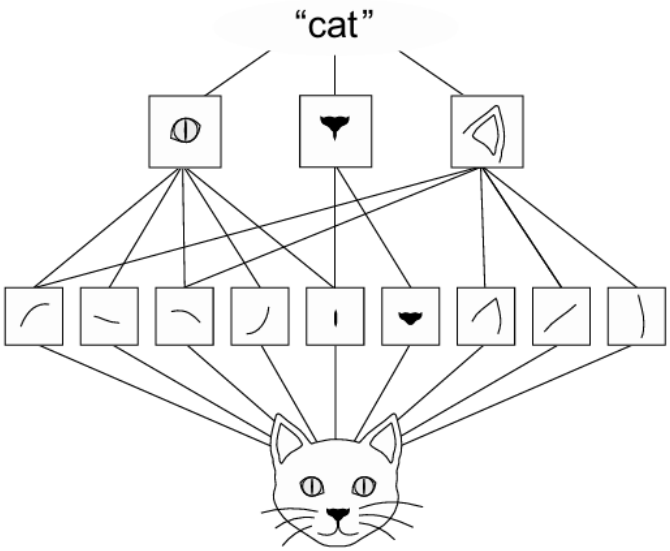

Figura 3.7: Esquema de los patrones espacialmente jerarquizados [17]

3.3. Autoencoders

Los autoencoders son una clase de redes neuronales no supervisadas que tienen como objetivo copiar la información de la entrada a su salida. Son redes entrenadas para reconstruir datos tras haber sido codificados previamente [26]. Poseen una parte *encoder*, que codifica la información en una capa intermedia o *hidden layer*, y otra *decoder* que realiza la reconstrucción. Ambas partes, habitualmente, representan funciones no lineales [24]. La capa intermedia, con menor número de neuronas, se conoce como cuello de botella o *bottleneck*. Esta estructura fuerza al modelo a crear un espacio latente de menor dimensión que la entrada y con ello captura las características más destacadas de los datos de entrenamiento [16]. El objetivo del *decoder* es reconstruir la salida lo más fielmente posible respecto de la entrada. La diferencia entre la entrada y la salida puede ser evaluada mediante una función de pérdida [24].

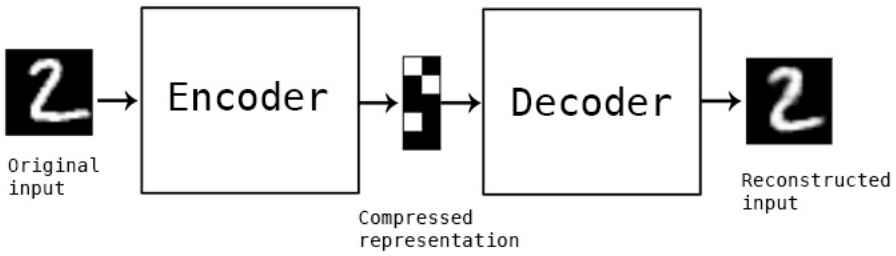

Figura 3.8: Esquema de un autoencoder [27]

Para este trabajo se ha optado por el uso de un autoencoder convolucional. Los autoencoders convolucionales suelen incluir capas de *pooling* o *strides* mayores que 1 para ir reduciendo el tamaño de la entrada en el *encoder*, así como el uso de capas con sobremuestreo o capas que usan la convolución transpuesta en la parte *decoder* para aumentar el tamaño del espacio latente en la reconstrucción *[23]* [26]. Es por esto último por lo que resultan interesantes para el objeto de estudio de este proyecto. Se puede, con ello, obtener una representación comprimida de los datos de entrada y posteriormente una posible reconstrucción de los mismos. Experimentalmente, los autoencoders profundos ofrecen una mejor compresión que aquellos más superficiales o lineales [16] [28].

Algunos de las principales áreas de aplicación de los autoencoders incluyen la detección de *outliers*, compresión de datos (objeto de nuestro experimiento), autoencoders para la eliminación de ruido o realce de imágnes, autoencoders VAE (variational autoencoders) y autoencoders GAN (generative adversarial autoencoder networks) para la generación de datos como imágenes o texto [26].

Capítulo 4

Metodología

En este capítulo se describen los procedimientos metodológicos seguidos para llevar a cabo el estudio sobre el uso de autoencoders como método de compresión de datos climáticos. Se detallarán las técnicas de recolección de datos, las decisiones clave en el diseño del modelo autoencoder, el entorno de ejecución, las métricas de evaluación, y el proceso de compresión y reconstrucción.

4.1. Análisis del uso de Autoencoders

Se ha analizado el uso de los autoencoders como técnica de compresión con pérdidas sobre datos climáticos, considerando su aplicabilidad y eficacia en la reducción de dimensionalidad. Los autoencoders son redes neuronales no supervisadas diseñadas para aprender representaciones eficientes de los datos, lo que los hace adecuados para tareas de compresión.

4.2. Recolección de datos climáticos

Se recolectaron datos climáticos desde el NOAA Physical Sciences Laboratory (PSL), específicamente de la temperatura del aire tanto de la tropopausa como de la superficie procedentes del proyecto NCEP/NCAR Reanalysis 1. Estos datos se eligieron debido a su cobertura global y su relevancia para el estudio de patrones climáticos. Los datos abarcan el periodo de 1948 a la actualidad, con una cobertura espacial de 2.5 grados de latitud por 2.5 grados de longitud, formando mapas de temperatura de dimensión 144x73.

4.3. Diseño del modelo Autoencoder

4.3.1. Arquitectura Encoder

Se realizó un estudio de las posibles estructuras para el encoder con el objetivo de lograr una compresión eficiente y preservar los detalles importantes para la reconstrucción. Se evaluaron diferentes configuraciones de la capa de cuello de botella y técnicas de reducción de tamaño, buscando aquellas que optimizasen la calidad de la compresión y el rendimiento del modelo.

4.3.2. Arquitectura Decoder

Para el diseño del decoder, se investigaron y compararon diversas estrategias de sobremuestreo y reconstrucción de datos. El objetivo fue encontrar un enfoque que maximizara la precisión de la reconstrucción y la conservación de los detalles. Se tomaron en cuenta técnicas de interpolación y *skip-connections* entre capas de la red que podrían mejorar la preservación de detalles en el proceso de reconstrucción.

4.4. Ajuste de hiperparámetros del modelo Autoencoder

Se llevó a cabo un proceso de ajuste y modificación de los hiperparámetros del modelo de autoencoder, enfocándose en aspectos como las funciones de activación (ReLU, GELU, ELU, Sigmoid, Tanh), los optimizadores (Adam, AdamW, Lookahead, AdaBelief), y las funciones de pérdida (MSE, RMSE, funciones personalizadas). Este ajuste se realizó mediante una búsqueda heurística llevada a cabo por el autor del trabajo. Las decisiones sobre el ajuste de los hiperparámetros se basaron en la necesidad de balancear la precisión de la compresión y la calidad de la reconstrucción. Todas las comprobaciones se realizaron mediante conjuntos de validación, y posteriormente con datos de test completamente independientes del diseño y ajuste del modelo final.

4.5. Entorno de ejecución

El entorno en el que se ha desarrollado el modelo del experimento ha sido Kaggle [29], una plataforma para ciencia de datos y aprendizaje automático que ofrece diferentes funcionalidades a usuarios y desarrolladores. Algunas de las herramientas y servicios que ofrece Kaggle son:

- Conjuntos de datos (Datasets) públicos para que lo usuarios puedan explorar y utilizar para sus proyectos. Existen conjuntos de datos médicos, financieros, climáticos, botánicos, etc.
- Recursos para aprendizaje y una comunidad para que los usuarios puedan aprender y compartir sus impresiones y conocimientos.
- Cuadernos al estilo Jupyter Notebook [30] que permiten escribir y ejecutar código de los desarrolladores. Ofrecen unidades de cómputo con unas cuotas de tiempo semanales gratuitas. La unidad de cómputo GPU del entorno de ejecución de Kaggle que se utilizó para el experimento fue la GPU P100 de Nvidia.
- Competiciones con el fin de que los desarrolladores y usuarios puedan participar en desafíos organizados por entidades y empresas que ofrecen una recompensa económica. Esto permite trabajar con problemas del mundo real.

Dentro del entorno se hace uso del lenguaje de programación Python [31], el cual se ha consolidado como el lenguaje predominante en el ámbito del aprendizaje automático y el aprendizaje profundo. Alguno de los puntos fuertes que han permitido que esto sea así son su simplicidad y legibilidad, la comunidad de Python, que es una de las más grandes del mundo, su integración con otros ecosistemas (como Jupyter Notebooks o Kaggle) y sobre todo la gran colección de bibliotecas y frameworks que posibilitan el desarrollo de software. Las bibliotecas que se han utilizado para el desarrollo del experimento de este proyecto son:

- *Keras*: es una biblioteca de alto nivel escrita en Python y puede ejecutarse sobre Tensorflow entre otros frameworks. Fue concebida para probar y experimentar de manera rápida bajo un enfoque de simplicidad y facilidad de uso. Posee un diseño modular que incluye modelos y diferentes arquitecturas de red [32].
- *Tensorflow*: es un framework de código abierto desarrollado por Google para computación y aprendizaje automático. Permite a los desarrolladores construir y probar modelos desde cero. Otra característica es la capacidad de ejecutarse en CPUs, GPUs y TPUs (Tensor Processing Units). Cabe destacar su herramienta para la visualización y monitoreo del entrenamiento, TensorBoard [33].
- *Matplotlib*: es una biblioteca de visualización que permite generar gráficos y figuras. En el proyecto se ha utilizado para generar las figuras de pérdidas, los histogramas de los errores o los mapas de temperatura [34].
- *Numpy*: es una biblioteca de alto nivel utilizada para generar vectores, matrices multidimensionales y permite el uso de herramientas matemáticas para procesar datos [35].

4.6. Métricas de evaluación

Para la evaluación del modelo desarrollado se ha tenido como objetivo minimizar el ratio de compresión (CR) de la codificación producida por la solución y superar el umbral de referencia de la correlación de Pearson (PCC) mencionado en [4], con ello se busca una mayor eficiencia en la compresión y una mayor calidad de la reconstrucción. Las métricas mencionadas anteriormente se definen como:

- Coeficiente de Correlación de Pearson (PCC): Mide la relación lineal entre la muestra original y la reconstruida.
- Coeficiente de Determinación (R^2): Indica la proporción de la varianza de los datos explicada por el modelo.
- Ratio de Compresión (CR): Calculado como el tamaño de la compresión dividido por el tamaño original.

4.7. Proceso de compresión y reconstrucción

Para el análisis del ratio de compresión (CR), se generaron dos archivos comprimidos: uno con los datos producidos por la predicción del encoder (codificación.7z) y otro con los residuos entre los valores originales y la predicción (errores.7z). Ambos archivos fueron convertidos a una precisión float16 e int16 para reducir el tamaño. La reconstrucción se realizó sumando la predicción del decoder a los residuos escalados, permitiendo evaluar el coeficiente de determinación y la calidad de la reconstrucción.

4.8. Entrenamiento y evaluación del modelo

El modelo de autoencoder convolucional fue evaluado en varias fases:

- Fase de entrenamiento: Se entrenó el modelo utilizando los datos climáticos de la tropopausa y la superficie desde 1948 hasta 2020, con un 20 % de los últimos datos temporales reservados para el conjunto de validación.
- Verificación del modelo con datos de 2021: Tras el entrenamiento, se evalúo el modelo con datos de 2021 que no fueron utilizados durante el entrenamiento y validación para asegurar su capacidad de generalización.
- Análisis de errores y residuos: Se estudiaron los errores y residuos del modelo, analizando el tamaño de compresión que éstos proporcionaban, así como visualizando su distribución a través de sus histogramas.

Capítulo 5

Descripción del modelo de compresión y reconstrucción

A continuación se describe la arquitectura de las partes del autoencoder propuesto y los elementos que lo definen, así como las consideraciones técnicas empleadas en su desarrollo. También se detallan las fases del procedimiento de compresión y reconstrucción diseñado para el estudio y análisis de la compresión de los datos climáticos.

5.1. Arquitectura encoder

El esquema utilizado para la parte encoder puede verse en la figura 5.1. En el diseño del encoder, y dada la naturaleza del problema, se ha tenido especial consideración en el tamaño del cuello de botella de la red. Tratándose de un problema de compresión, es deseable que esta capa final del encoder tenga unas dimensiones lo más ajustadas posibles para una mayor compresión de los datos, reduciendo así el futuro tamaño de los archivos. Se ha optado por un diseño con una capa cuello de botella de dimensiones mayores (en altura y anchura), en detrimento de los canales utilizados en ella, siguiendo el trabajo de [26], donde concluyen que el uso de un cuello de botella de mayor tamaño puede tener un impacto positivo en las redes convolucionales reduciendo los errores en el entrenamiento, acelerando el tiempo de entrenamiento y preservando mayores detalles en la imagen para una mejor reconstrucción. Es por ello que en este trabajo el cuello de botella de la red consta de un tamaño de 32x32x1.

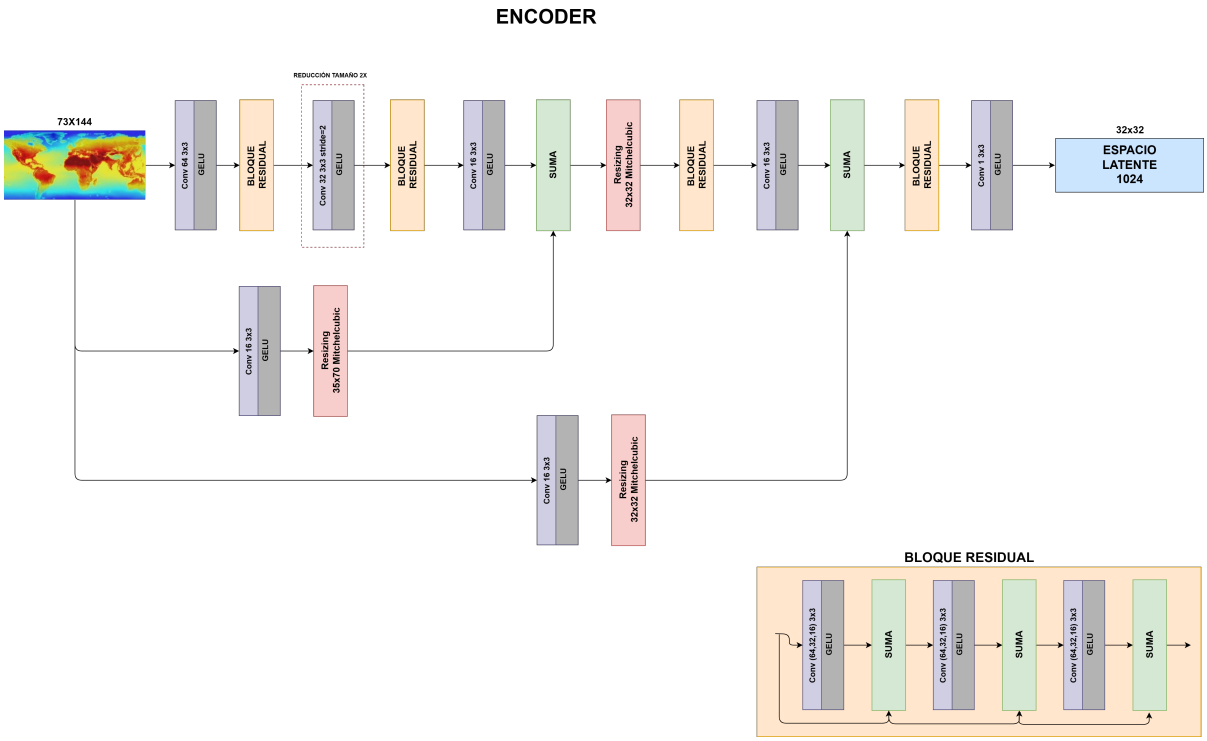

Figura 5.1: Esquema de la parte encoder del modelo utilizado

La segunda consideración adoptada para la construcción del modelo de encoder ha sido la forma en la que la red disminuye el tamaño de los datos a medida que éstos se aproximan al cuello de botella. En muchos trabajos se opta por muestrear los datos usando Max Pooling en las capas de la red, esto consiste en elegir el píxel de mayor valor de entre un conjunto específico de píxeles, desechando el resto. Con ello es posible que perdamos información útil en el proceso. En este trabajo se ha seguido un enfoque similar al adoptado en [36] para imágenes, utilizando capas *resize* para muestrear los datos y reducir su tamaño. En [36] mencionan como esta solución puede resultar útil en modelos encoder-decoder y aprovechar estos elementos *resize* tanto para muestrear como para sobremuestrear datos en la parte decoder como sustitución de las capas de *Upsampling*, ampliamente utilizadas.

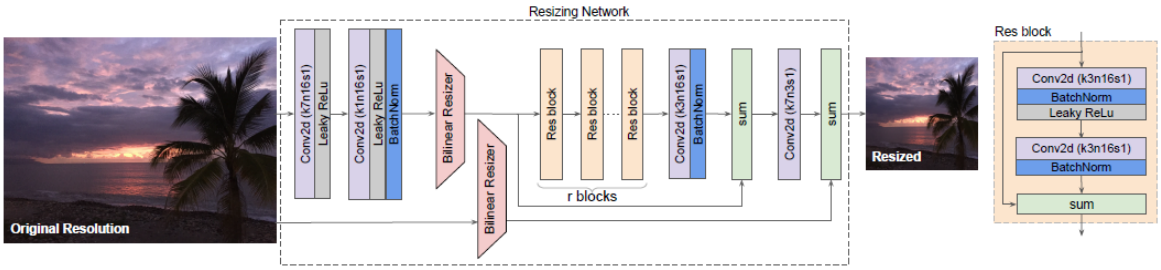

Figura 5.2: Modelo de CNN para reescalar imágenes propuesto en [36]

El uso de algoritmos que tengan un buen desempeño en el muestreo a menudo hace

que no se comporten tan bien en el sobremuestreo [37]. A diferencia del trabajo mostrado en [36], se ha usado el algoritmo MitchellCubic en las capas de muestreo y Lanczos3 para las de sobremuestreo en la parte decoder.En [37] parecen ser los que mejor desempeño muestran para ambas tareas.

Las *skip-layer connections* son un método a menudo utilizado en modelos de redes convolucionales ResNet, MobileNet o EfficienteNet. Consiste en unir o conectar capas convolucionales con otras más profundas, con ello se consigue pasar información de la imagen, como detalles, a otras partes de la red, como el decoder en los casos de redes que mitigan el ruido [38]. Esto permite que el entrenamiento de la red pueda converger más rápido y mitigar en parte el desvanecimiento del gradiente [39].

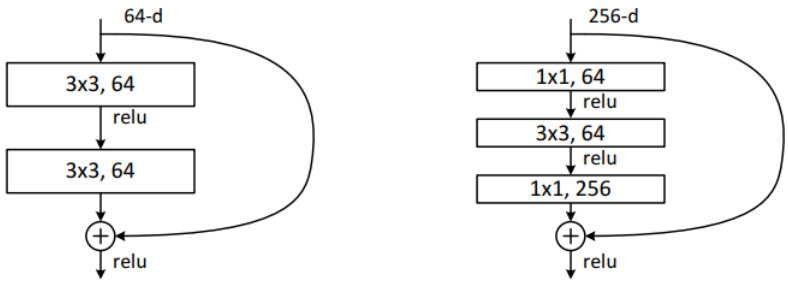

Figura 5.3: Ejemplo de skip connection en bloques de redes ResNets [40]

Para las capas convolucionales de la parte encoder se ha decidido utilizar una función de activación GELU (Gaussian Error Linear Unit). Es una función de activación reciente y combina propiedades lineales y no lineales. Algunas de las características más importantes son: una forma de función más suave y continua, a diferencia de las funciones de activación ReLU (Rectified Linear Unit) y una mejora en el rendimiento en tareas como procesamiento del lenguaje natural y la visión artificial [41]

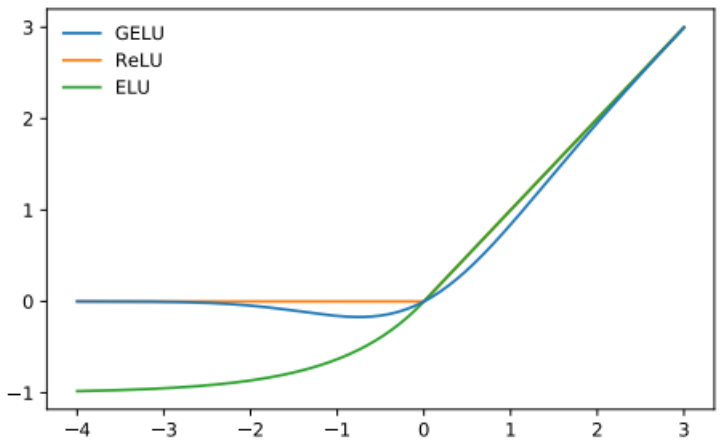

Figura 5.4: Ejemplo de las funciones de activación GELU, ReLU y ELU [41]

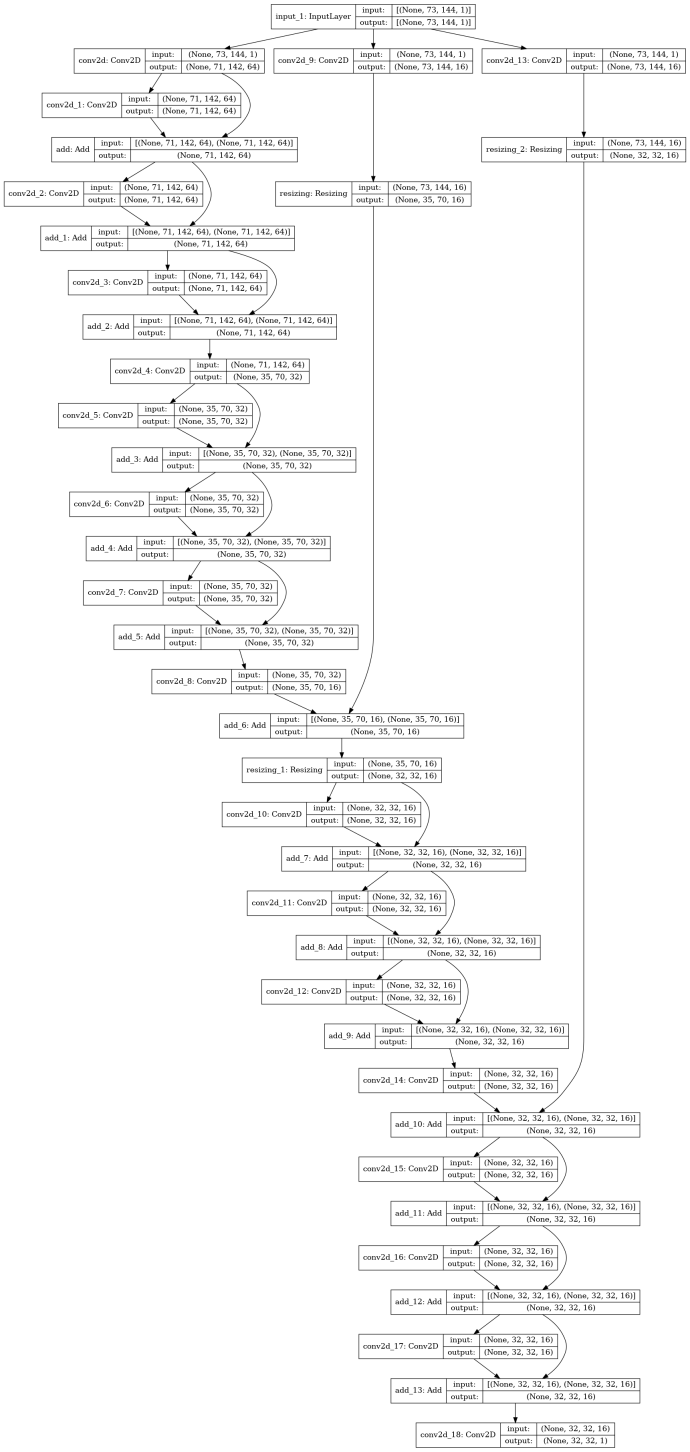

Figura 5.5: Sumario en Keras de la arquitectura de la parte encoder del modelo

5.2. Arquitectura decoder

La parte decoder del modelo autoencoder se ha diseñado de una manera similar a la utilizada en la parte encoder, utilizando capas *Resizing* para sobremuestrear los datos de entrada al estilo de lo utilizado en el trabajo [42]. La figura 5.6 representa un esquema

de la arquitectura del decoder. Como se comentó en el apartado de descripción de la arquitectura encoder, se ha optado por utilizar una interpolación Lanczos3, la cual podría comportarse mejor en tareas de interpolación. También se han empleado capas convolucionales transpuestas 2D para el proceso de reconstrucción y una sola *skip connection* que hace una operación de *resize* al tamaño objetivo de la reconstrucción (73x144) y conecta la capa de entrada del decoder con una capa aditiva.

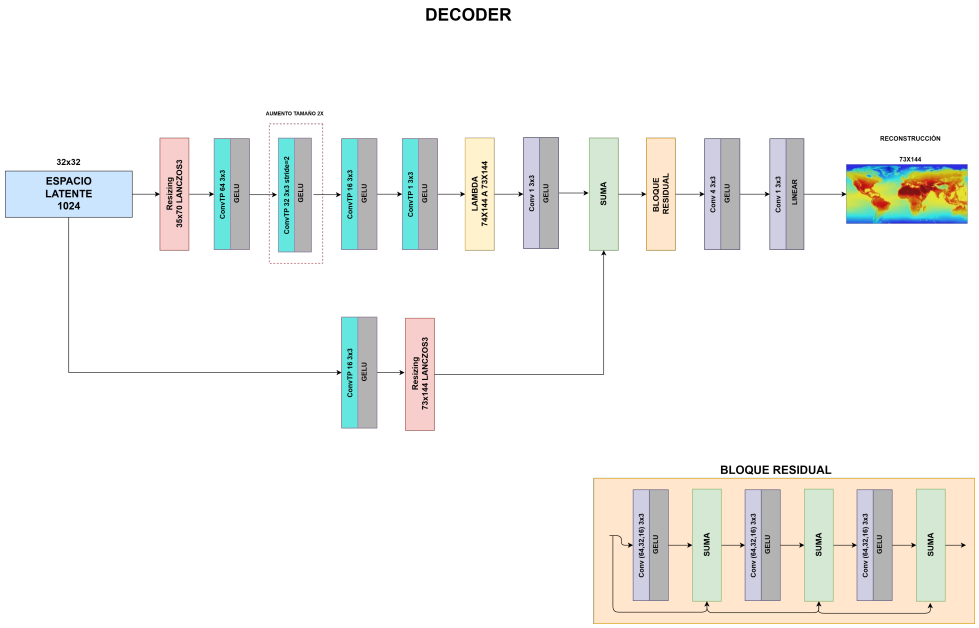

Figura 5.6: Esquema de la parte decoder del modelo utilizado

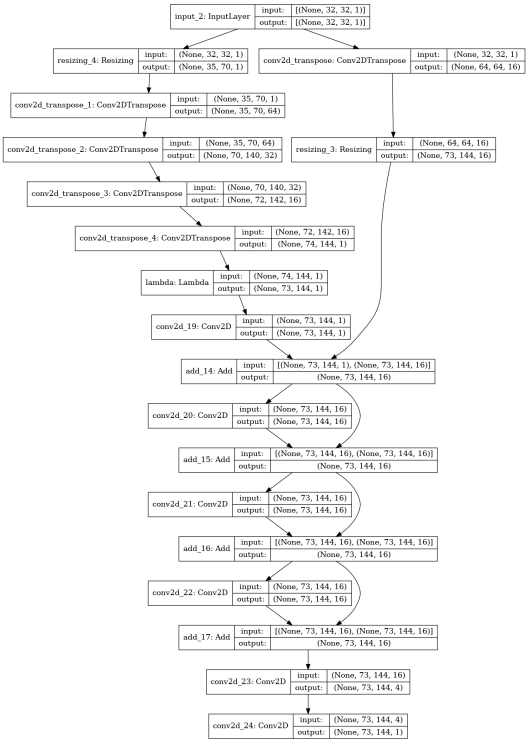

Figura 5.7: Sumario en Keras de la arquitectura de la parte decoder del modelo

El modelo autoencoder cuenta con un total de 210.581 parámetros:

```
==========================
Total params: 210,581
Trainable params: 210,581
Non-trainable params: 0
```

Figura 5.8: Informe de Keras del número de parámetros del modelo autoencoder

5.3. Proceso de compresión

La parte encoder del modelo autoencoder anteriormente descrito recibe los datos de temperaturas como datos de entrada, produciendo como salida una representación comprimida de los datos en el espacio latente generado. Estos datos comprimidos tienen una dimensión de 32x32x1, lo que resulta en una representación de tamaño 1024 sobre unos datos de entrada de dimensión 73x144x1, con tamaño 10512. Este espacio latente de datos de la compresión se convierte a formato float-16 proveniente de una precisión float-32, y tras ello se utiliza un compresor que comprime sin pérdida estos datos a un formato 7z, obteniendo el archivo *codificación.7z* de la figura 5.9.

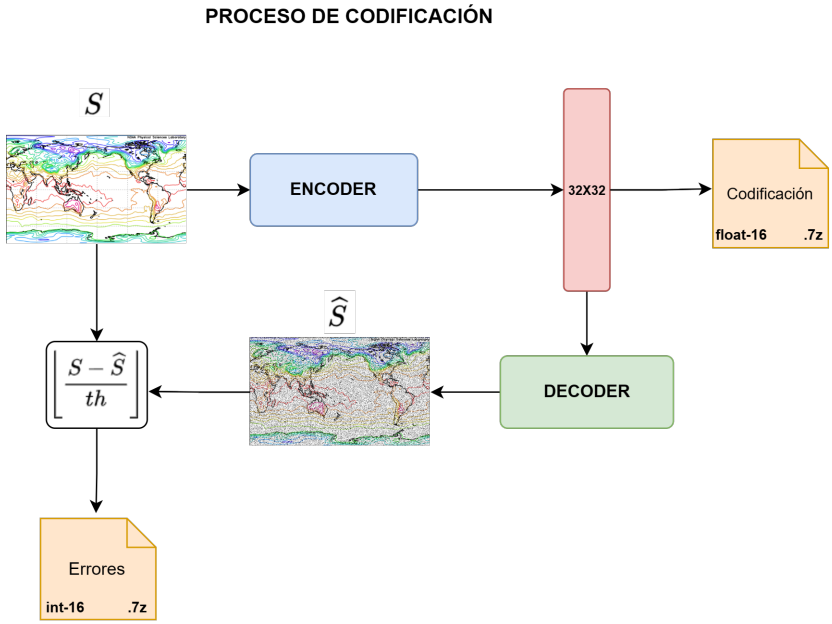

Figura 5.9: Esquema de la compresión

Por otro lado, la representación comprimida de los datos de salida de la parte encoder del modelo corresponden a los datos de entrada de la parte decoder del autoencoder. La salida del decoder produce la reconstrucción de los datos comprimidos $\hat{S}$. Con los datos de la muestra original S y los datos de la predicción del modelo autoencoder $\hat{S}$, se calculan

los residuos mediante la diferencia de la muestra original y su predicción, escaladas por un factor de *threshold*, cuyo objetivo es acotar el error. El valor de *threshold* durante el proyecto se fijó en 0,004. Posteriormente, se realiza un redondeo a número entero de estos residuos y se guardan como int-16 para conseguir una reducción del tamaño del archivo. Finalmente este archivo se comprime sin pérdidas a formato 7z y se obtiene el archivo *errores.7z*.

La idea detrás de este proceso de compresión es similar a la utilizada por fpzip para la codificación predictiva, donde los residuos entre la muestra original y la predicción de los valores en coma flotante es más pequeña que el valor original y por tanto puede ser codificada con un número menor de bits [7].

5.4. Proceso de descompresión y reconstrucción

Para llevar a cabo la reconstrucción de los datos comprimidos y, siguiendo un proceso inverso al realizado en el proceso de compresión, los datos procedentes de la compresión almacenados en el archivo *codificación.7z* son descomprimidos e introducidos, en formato float-16, a la parte decoder como entrada. Como se vió en el apartado anterior, con los datos de compresión, el decoder produce como salida la predicción $\hat{S}$. Así mismo, el archivo de errores descomprimido *errores.7z*, con precisión int-16, es multiplicado por el factor de *threshold* y, sumando la predicción $\hat{S}$, se obtiene la muestra reconstruida. La figura 5.10 muestra un esquema de este proceso.

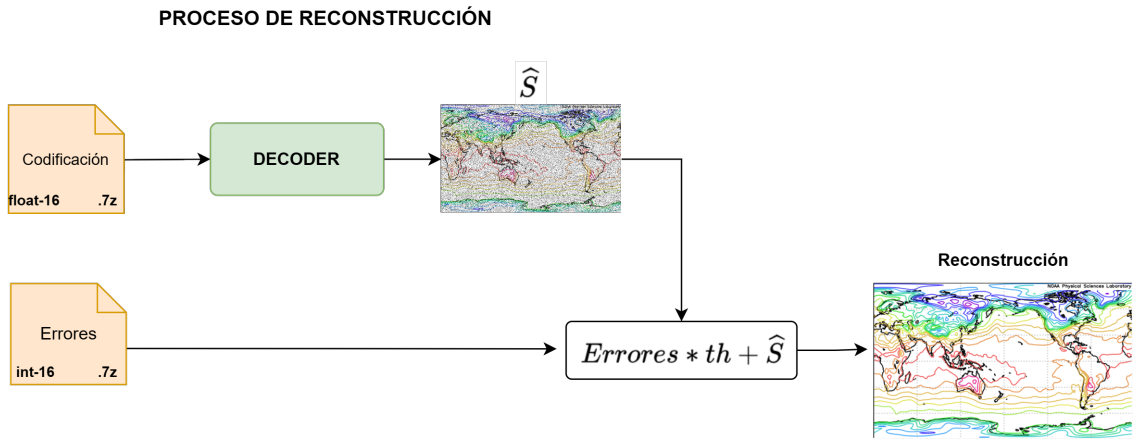

Figura 5.10: Esquema de la reconstrucción

Los datos obtenidos como resultado de ambos procesos permiten realizar el análisis de la solución a estudiar en este proyecto. El objeto de estudio de este trabajo es encontrar una solución que, atendiendo a la calidad de la reconstrucción de los datos comprimidos, supere un determinado umbral de referencia y permita minimizar lo máximo posible el

tamaño de la compresión de los datos, utilizando como métrica el ratio de compresión (CR) entre los datos comprimidos y la muestra original de los mismos.

$$CR = \frac{size(compressed)}{size(original)} = \frac{size(errores,7z+codificacion,7z)}{size(original)}$$

Capítulo 6

Caso de estudio

6.1. Datos climáticos

Como conjunto de entrenamiento y validación, se han utilizado los datos climáticos relativos a la temperatura del aire de la tropopausa [43] y la superficie [44] recogidos en NCEP/NCAR Reanalysis 1 [45], la figura 6.1 muestra un ejemplo del mapa de temperatura de los dos datasets. Se ha optado por utilizar dos conjuntos de datos para poder comprobar y evaluar si los resultados obtenidos por el modelo utilizado eran consistentes. Las valores de temperatura de los datasets tienen las siguientes características:

- Los conjuntos de datos contienen valores diarios de temperatura desde 1948 hasta la actualidad.
- Cobertura espacial de 2.5 grados latitud x 2.5 grados longitud globales (dimensión de los mapas de temperatura de 144x73). 90N - 90S, 0E - 357.5E
- Para los conjuntos de entrenamiento se han utilizado las medidas desde 1948 hasta 2020, el tamaño de cada mapa de temperaturas es de 144x73.
- Para el conjunto de validación del entrenamiento se ha utilizado el 20 % de los últimos datos temporales del conjunto de entrenamiento.
- Verificación posterior al entrenamiento mediante un conjunto de datos de test formados por los valores obtenidos en el 2021 exclusivamente.

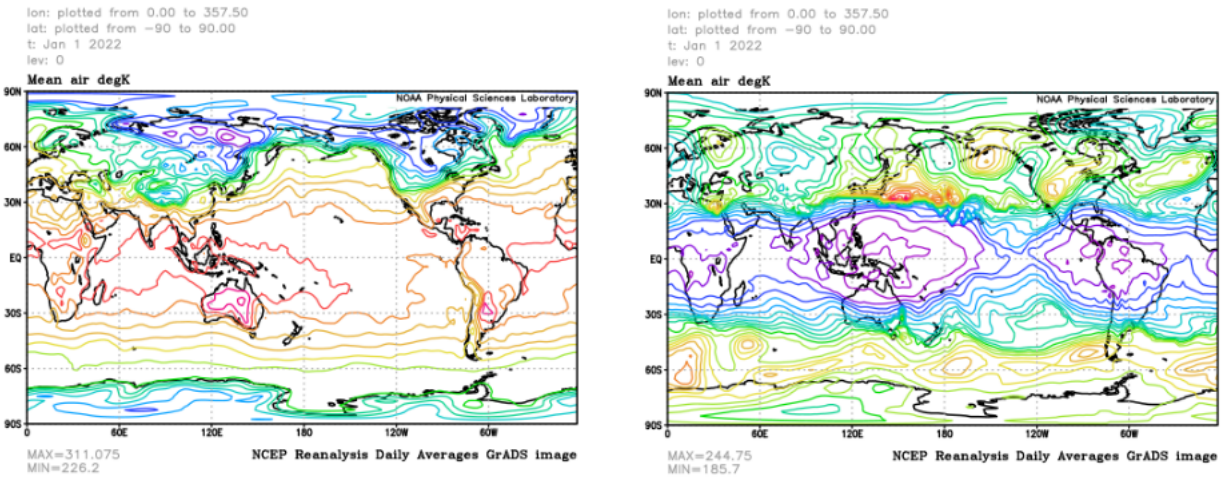

Figura 6.1: a)mapa datos temperatura aire en la superficie, b)mapa datos temperatura aire tropopausa

La tabla 6.1 detalla el número de muestras de los conjuntos de Train, validación y Test utilizados para el entrenamiento y verificación.

	Train	Validación	Test(2021)
Tropopausa	26348	20 % train	275
Superficie	26664	20 % train	334

Tabla 6.1: Número de muestras de los conjuntos de datos utilizados

6.1.1. Normalización de los datos

Con el fin de centrar los datos y acotarlos, se ha aplicado la siguiente normalización sobre todos los conjuntos de datos utilizados:

$$X_{entrenamiento} = \frac{(X_{entrenamiento} - \mu)}{\text{máx}|X_{entrenamiento} - \mu|} \ ,$$

donde μ es la media de los datos de entrenamiento.

6.2. Métricas

En los trabajos de Baker et al. [4] [7] utilizan el coeficiente de correlación de Pearson (PCC) como parámetro para evaluar la calidad de la reconstrucción de la compresión con pérdidas. El PCC indica el grado de relación lineal entre la muestra original y la reconstruida, indicando un valor de 1 una reconstrucción perfecta. El PCC viene determinado por:

$$\rho = \frac{cov(X,\widetilde{X})}{\sigma_X \sigma_{\widetilde{X}}},$$

donde cov(X,$\widetilde{X}$)es la covarianza y $\rho \in [-1, 1]$. Sería deseable que la compresión con pérdidas no degradase esta relación en demasía para un posterior análisis de los datos reconstruidos. En esto trabajos mencionados utilizan un valor PCC $\geq$ 0,99999 como umbral aceptable para la evaluación.

Para la realización de esta prueba también se ha tomado este valor como referencia, utilizando como métrica de evaluación el coeficiente de correlacion R^2, que viene determinado por:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y})^2}{\sum_i (y_i - \bar{y})^2} = \rho^2 \ ,$$

por tanto, para nuestro estudio, el valor de referencia umbral es $\rho^2 \geq R^2 \geq 0,9999800001$. También se ha utilizado como métrica el tamaño de compresión (CR):

$$\text{CR} = \frac{tamañoCompresión}{tamañoOriginal} \ ,$$

6.3. Condiciones de entrenamiento

En esta sección se describen los elementos utilizados en el proceso de entrenamiento del modelo experimento. Se mencionan también aquellos elementos o técnicas descartadas que han sido probadas y evaluadas durante el desarrollo del proyecto.

6.3.1. Optimizadores

Los optimizadores son algoritmos que ajustan los parámetros del modelo con el fin de minimizar la función de pérdida. Hacen uso de la tasa de aprendizaje y el gradiente calculado mediante la función de pérdida. Durante el desarrollo y evaluación del modelo se han utilizado distintos optimizadores, como son Adam [46], AdamW [47], AdamRectified [48], AdaBelief [49] y los optimizadores basados en una tasa de aprendizaje cíclica (CLR) [50].

Como optimizador inicial y referencia, se utilizó Adam (Adaptive Moment Estimation). Es un optimizador utilizado frecuentemente en el entrenamiento de redes neuronales profundas, combinando ventajas de otros optimizadores como AdaGrad y RMSProp. Sus principales características son: una tasa de aprendizaje adaptativa y su eficiencia computacional y uso de memoria. También se evaluó el rendimiento con otros optimizadores de la familia Adam, como AdamW (Adam Weight), una variante del optimizador Adam

que desacopla la regularización o decaimiento de los pesos del proceso de optimización de la función de pérdidas. En [47] mencionan un beneficio sobre Adam en la capacidad de generalización y permitiendo competir con SGD en tareas de clasificación.

El enfoque del trabajo [48] se utilizó como marco para la obtención de una combinación de técnicas que pudieran ser analizadas con Adam como referencia. En este trabajo de I.Loshchilov y F.Hutter presentan el optimizador Ranger21, basado en una combinación de técnicas entre las que destaca Lookahead [51] y el uso de AdamW. Lookahead es un tipo de algoritmo de optimización diferente a los de tasa de aprendizaje adaptativa y basados en el momentum, consiste en la elección de una dirección de búsqueda *hacia el futuro* en la secuencia de *pesos rápidos* generados por otro optimizador. Ésto, según los autores, permite una mejora en la estabilidad del aprendizaje y una mejora del rendimiento de Adam sobre datasets como ImageNet y CIFAR10/100 [51]. Previamente, para el desarrollo de Ranger, utilizaron AdamRectified como segundo optimizador.

Para el presente trabajo, se utilizó una combinación de Lookahead junto con AdaBelief como segundo optimizador emulando el funcionamiento del optimizador Ranger mencionado anteriormente. El optimizador AdaBelief pretende alcanzar tres metas: una rápida convergencia al igual que los optimizadores adaptativos, una buena generalización propia de SGD y estabilidad en el entrenamiento [49]. AdaBelief adapta el tamaño de la tasa de aprendizaje según una *creencia* (*belief*) sobre la dirección del gradiente. A raíz de la observación de la media móvil exponencial (EMA) del gradiente como predicción, puede generar un paso en la tasa de aprendizaje más bajo si esta predicción no es cercana al gradiente observado; y una tasa de aprendizaje más grande si esta predicción es cercana a la observación. Según los autores [49], AdaBelief mejora significamente a otros métodos en tareas de clasificación de imágenes, especialmente sobre ImageNet, y modelos de lenguaje.

Por último, tras evaluar su desempeño, se descartaron los optimizadores basados en una tasa de aprendizaje cíclica, en los cuales la tasa de aprendizaje varía entre un máximo y un mínimo según una determinada política. La figura 6.2 ilustra una tasa de aprendizaje que varía de forma cíclica triangular.

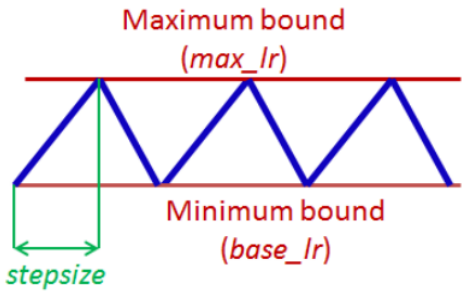

Figura 6.2: Política CLR de tasa de aprendizaje trinagular [50]

Tanto Lookahead, como Adabelief o las diferentes variantes de Adam, fueron evaluadas tanto por separado como en una combinación siguiendo el marco de Ranger. Finalmente, para la evaluación de los resultados del modelo se utilizaron dos optimizadores: Adam y la combinación Lookahead junto con AdaBelief. La tasa de aprendizaje inicial se estableció en $1 * 10^{-3}$.

6.3.2. Funciones de pérdida

Las funciones de pérdida miden la diferencia entre las predicciones del modelo y las etiquetas o muestras reales. De forma similar al análisis de los optimizadores, se utilizaron y consideraron varias funciones de pérdida para este proyecto. Al tratarse de una tarea de compresión y reconstrucción de imágenes, se utilizaron funciones de pérdida de regresión, como son: el error cuadrático medio (MSE), el error absoluto medio (MAE) o la raíz del error cuadrático medio (RMSE).

El error cuadrático medio, queda caracterizado por:

$$mse = \frac{1}{n}\sum_{i=0}^{n}(\hat{Y}_i - Y_i)^2$$

Mientras el error medio absoluto (MAE) viene determinado mediante:

$$mae = \frac{1}{n}\sum_{i=0}^{n}|\hat{Y}_i - Y_i|$$

La principal diferencia entre ambos es que el MSE es más sensible a los valores atípicos (*outliers*) que el MAE, ya que éstos se elevan al cuadrado. Así, el MSE se utiliza preferiblemente en modelos cuyo objetivo sea minimizar grandes errores, penalizando grandes desvíos. También, cabe destacar que la derivada del MSE es una función continua y suave, no siendo así en el caso del MAE. Como función de pérdidas para el modelo experimento, en concreto, se utilizó el MSE, además de la raíz cuadrada del MSE (RMSE) siguiendo los trabajos de Baker et. al [4], en el que lo utilizan como métrica en su metodología.

Para el experimento se ha elaborado una función de pérdidas propia, denominada en el proyecto como *Custom*. La función se ha desarrollado según el enfoque seguido en [52], donde los autores exploran el uso de una función de pérdidas en el dominio de la frecuencia para compresión de imágenes. En la figura 6.3 se puede ver un esquema de la arquitectura empleada en el trabajo mencionado.

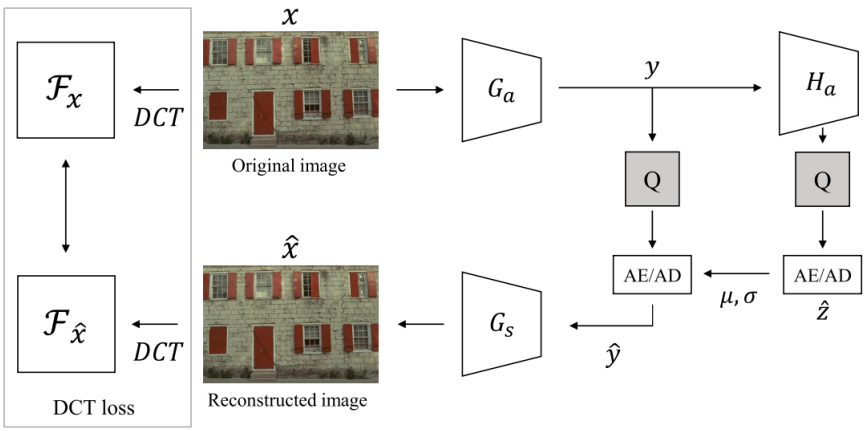

Figura 6.3: Arquitectura de red neuronal para compresión de imágenes con función de pérdidas basadas en DCT [52]

En el trabajo [52] utilizan la transformada del coseno discreta (DCT) para convertir las muestras reales y de la predicción al dominio de la frecuencia y después calcular el MSE con ellas. En este proyecto se ha procedido de manera similar y para definir la función *Custom* se han calculado las DCT de las predicciones y las muestras reales para después obtener el RMSE. Así, finalmente, se han evaluado los resultados del MSE, el RMSE y la función *Custom* sobre los diferentes optimizadores utilizados.

Por último, cabe mencionar que para el experimento se exploró el uso de otras funciones *Custom* similares a las empleadas en [53], en donde se evalúa el desempeño del uso de los índices de similaridad estructural (SSIM) y su versión multi-escala (MS-SSIM) como funciones de pérdida en tareas de reconstrucción de imágenes. Estas funciones fueron descartadas para la evaluación final del modelo experimento ya que en las pruebas preliminares tuvieron un peor desempeño.

Capítulo 7

Resultados del caso de estudio

El estudio de los resultados se centra en los dos conjuntos de datos de temperatura descritos en la sección 6.1 bajo las condiciones de entrenamiento detalladas en la sección 6.3. Primero se mostrará el resultado de la fase de entrenamiento del modelo sobre el conjunto de datos de la temperatura de la tropopausa y una posterior evaluación adicional con datos del 2021 no utilizados durante la fase de entrenamiento. Posteriormente, se procederá de igual manera con los datos del conjunto de la temperatura de la superficie.

7.1. Conjunto de datos de la temperatura en la tropopausa

7.1.1. Fase de entrenamiento:

El modelo autoencoder solución se ha probado con dos optimizadores diferentes, Adam y la combinación de la técnica Lookahead con el optimizador AdaBelief. Ambos optimizadores han teniendo como función de pérdidas el MSE, RMSE y la función *Custom* definida anteriormente. La métrica utilizada para evaluar la calidad del modelo ha sido el coeficiente de determinación R^2. La tabla 7.1 muestra el coeficiente de determinación del conjunto de validación obtenido tras 600 épocas de entrenamiento.

	MSE	RMSE	CUSTOM
ADAM	-	(0.9964 it.328)	0.9962
LOOKAHEAD+ADABELIEF	-	0.9964 - 0.9965	0.9964 - 0.9965

Tabla 7.1: Coeficiente de determinación del conjunto de validación tras 600 épocas del modelo bajo los optimizadores ADAM y LK+AdaBelief

El entrenamiento de ambos optimizadores no llegó a término para el caso de la función de pérdida MSE. Tanto el entrenamiento con RMSE como el realizado con la función *Custom* tuvo resultados similares en cuanto al coeficiente de determinación. El entrenamiento con el optimizador Adam con función de pérdida RMSE se detuvo en la iteración 328. A continuación se pueden ver las gráficas de errores de entrenamiento y validación de las combinaciones que llegaron a término sobre el conjunto de datos de la tropopausa.

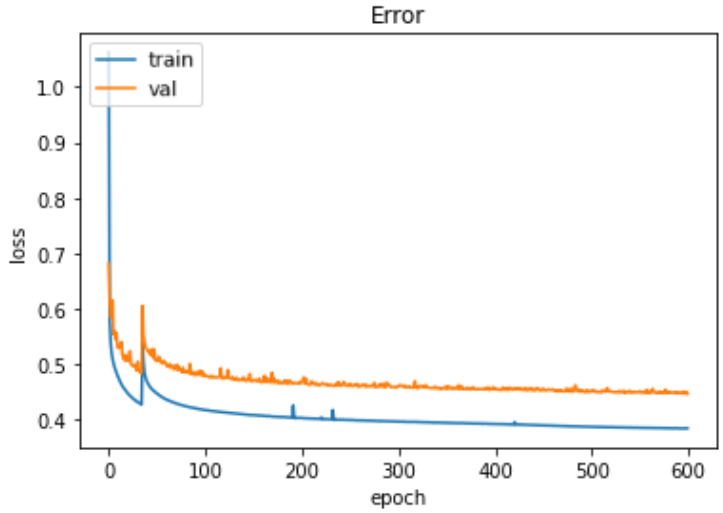

Figura 7.1: Gráfica error train vs.val adam-custom

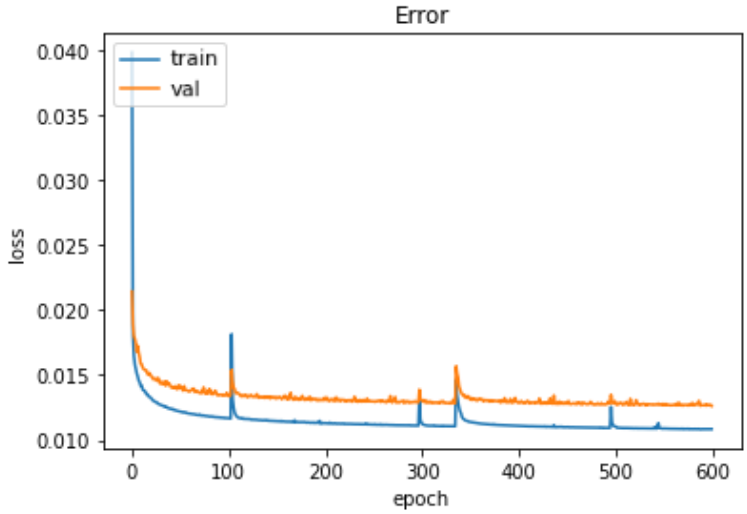

Figura 7.2: Gráfica error train vs. val Lookahead+Adabelief - RMSE

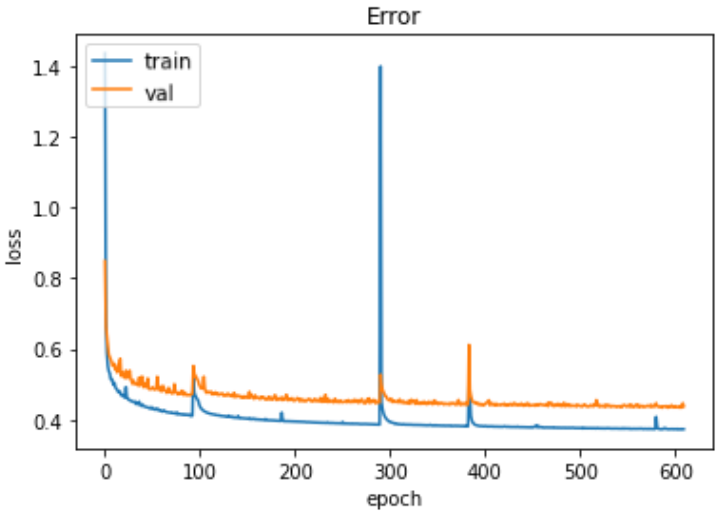

Figura 7.3: Gráfica error train vs. val Lookahead+Adabelief - Custom

Se aprecia como hubo algunas inconsistencias durante el entrenamiento en todos los modelos, siendo más significativas en la combinación LK+AdaBelief con la función de pérdidas *Custom*.

7.1.2. Evaluación con el conjunto de datos de 2021

Tras el entrenamiento, se comprobaron las combinaciones que llegaron a término en la fase anterior con datos de la tropopausa extraídos exclusivamente del año 2021 y no utilizados para el entrenamiento. Siguiendo el procedimiento de compresión y reconstrucción descrito en el **capítulo 5**, se calcularon los coeficientes de determinación de las configuraciones evaluadas.

	MSE	RMSE	CUSTOM
ADAM	-	-	0.999980274
LOOKAHEAD+ADABELIEF	-	0.999980288	0.999980279

Tabla 7.2: Coeficiente de determinación del conjunto de test del modelo bajo los optimizadores ADAM y LK+AdaBelief sobre el conjunto de datos de la tropopausa de 2021

En la tabla 7.2 se observa como el coeficiente de determinación del modelo supera, en sus diferentes variantes, el umbral aceptable objetivo descrito en las métricas de la **sección 7.4** $\rho^2 \geq R^2 \geq 0{,}9999800001$

7.1.2.1. Ratio de la compresión (CR)

Siguiendo el proceso de compresión descrito en el **capítulo 5**, se obtienen los archivos de codificación y errores tras computar la predicción del modelo con el objetivo de calcular

el ratio de compresión de cada una de las combinaciones. Para ello, utilizamos la métrica

$$\text{CR}=\frac{tamañoCompresión}{tamañoOriginal} ,$$

donde la parte de la compresión está determinada por la suma del archivo de codificación procedente de la predicción de la parte encoder y del archivo de errores procedente de los residuos entre la predicción de la parte decoder y las muestras originales. En la tabla 7.3 se muestran los resultados del ratio de compresión obtenido.

	MSE	RMSE	CUSTOM
ADAM	-	-	0,33360
LOOKAHEAD+ADABELIEF	-	0,33906	0,34109

Tabla 7.3: Ratio de compresión CR obtenido para las distintas combinaciones de optimizadores y funciones de pérdida sobre los datos comprimidos del conjunto de test de 2021

No se aprecian diferencias significativas entre las distintas combinaciones, siendo el optimizador Adam junto a la función de pérdidas *Custom* el que mejor desempeño tuvo.

7.1.2.2. Histograma de los residuos

La distribución de los residuos de las diferentes combinaciones del modelo sobre los datos del conjunto de evaluación del 2021 puede verse en los siguientes histogramas.

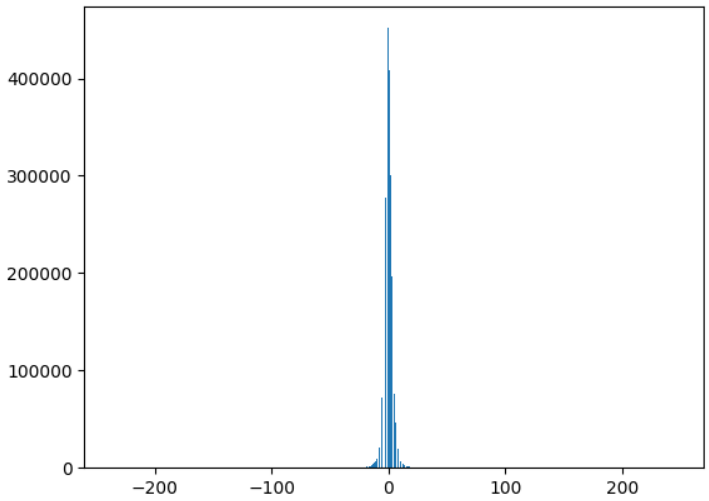

Figura 7.4: Histograma de los residuos adam-custom

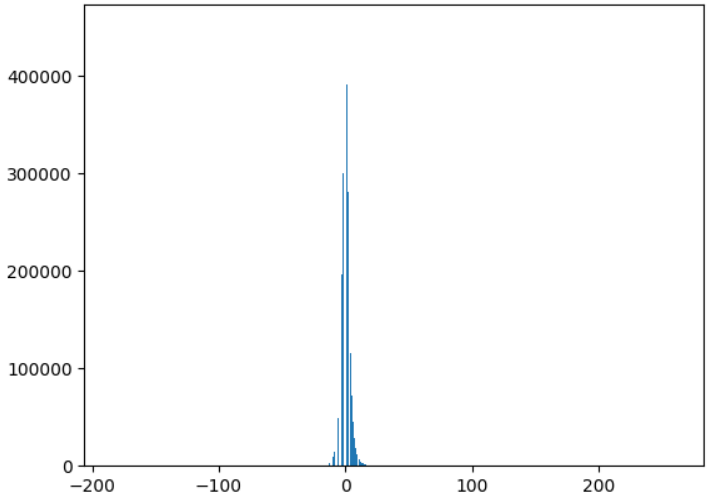

Figura 7.5: Histograma residuos Lookahead+Adabelief - RMSE

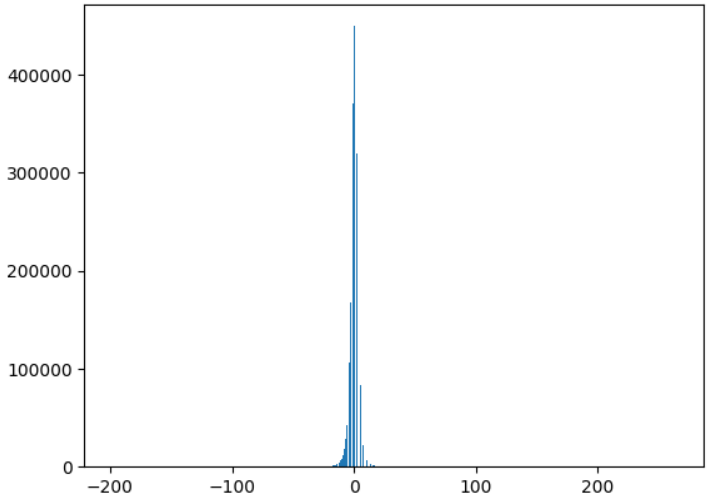

Figura 7.6: Histograma de los residuos Lookahead+Adabelief - Custom

7.2. Conjunto de datos de la temperatura en la superficie

Tras las pruebas del modelo experimento sobre los datos del conjunto de la temperatura de la tropopausa, se decidió analizar el desempeño de dicho modelo sobre otro conjunto de datos de temperatura de dimensiones similares. El conjunto elegido fue el de la temperatura en la superficie descrito en el **capítulo 6.1**. Se utilizó el modelo experimento junto con la combinación del optimizador Lookahead+AdaBelief y la RMSE como función de pérdida ya que esta configuración obtuvo un ligero mejor coeficiente de determinación y un ratio de compresión similar a las demás combinaciones. El resultado del coeficiente de determinación tras la fase de entrenamiento puede verse en la tabla 7.4.

	RMSE
LOOKAHEAD+ADABELIEF	0.9990

Tabla 7.4: Coeficiente de determinación del conjunto de validación del modelo elegido sobre los datos de la temperatura en la superficie

El coeficiente de determinación fue considerablemente mejor que el obtenido durante el entrenamiento con el conjunto de datos de la tropopausa debido, posiblemente, a una naturaleza más suave de los datos de temperatura del conjunto de datos de la superficie.

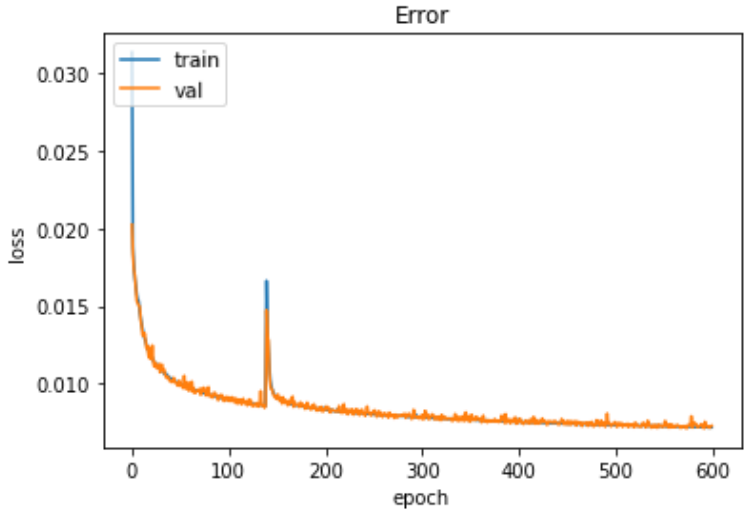

Figura 7.7: Gráfica error train vs. val Lookahead+Adabelief - RMSE

La gráfica 7.7 indica un buen desempeño del modelo en cuanto a la generalización de los datos de validación y tamaño de los errores.

7.2.1. Evaluación con el conjunto de datos de 2021

Al igual que se hizo con el conjunto de datos de la tropopausa, se evaluó el modelo entrenado con un conjunto de test no utilizado de datos proveniente del año 2021. Siguiendo el procedimiento de compresión y reconstrucción descrito en el **capítulo 5**, se obtuvieron las siguientes métricas:

El coeficiente de determinación fue igual a 0.9999806058333789, superando así el valor umbral aceptable objetivo en el presente trabajo y descrito en la **sección 6.4** $\rho^2 \geq R^2 \geq 0{,}9999800001$

El ratio de la compresión(CR) entre el archivo de codificación y los residuos de la predicción del modelo respecto del tamaño original del archivo de temperaturas fue de

$$CR = \frac{tamañoCompresión}{tamañoOriginal} = (1,14MB + 525KB)/6,208KB = 0,2738402061$$

Las tablas 2.2, 2.3 y 2.4 mencionadas en el **apartado 2** correspondiente al estado del arte, muestran los ratios de compresión obtenidos en los trabajos de Baker 2014, 2016 y 2017.

	GRIB2	*ISABELA*	*fpzip*	*APAX*	NC
avg. CR	0.37	0.42	0.18	0.29	0.61
best CR	0.03	0.20	0.02	0.06	0.07
worst CR	0.86	0.77	0.68	0.80	0.86
avg. ρ	.9999999	.9999991	.9999995	.9999991	1.0
avg. nrmse	5.73e-5	3.22e-4	2.35e-4	2.61e-4	0.0
avg. e_{nmax}	1.01e-4	5.56e-3	2.76e-3	1.83e-3	0.0

Tabla 7.5: Tabla con los resultados de compresión de los algoritmos analizados en [4]

Method	Monthly	Daily	6-hourly	Average
fpzip	.15	.22	.18	.18
NetCDF-4	.51	.70	.63	.62
Truncation	.61	.58	.60	.69

Tabla 7.6: Tabla con los ratios de compresión en diferentes frecuencias de recogida de los algoritmos analizados en [4]

variable name	*SPECK*				*fpzip*				DWT→IDWT
	variant	e_{nmax}	*nrmse*	*CR*	variant	e_{nmax}	*nrmse*	*CR*	max. abs. error
H2O2	speck_2	2.47e-4	2.47e-5	.06	fpzip_20	2.56e-4	2.05e-5	.23	0.0
FSNTC	speck_8	1.75e-4	2.08e-5	.26	fpzip_24	2.33e-5	1.18e-5	.36	0.0
TS	speck_4	1.46e-3	1.95e-4	.13	fpzip_24	8.71e-5	4.95e-5	.28	0.0
TAUY	speck_12	8.04e-6	1.24e-6	.38	fpzip_24	1.41e-5	7.92e-7	.54	0.0
CLOUD	–	–	–	–	fpzip_24	1.70e-5	2.42e-6	.36	8.88e-16
PRECSC	–	–	–	–	fpzip_16	4.01e-3	1.97e-4	.12	2.53e-24
TOT_ICLD_VISTAU	–	–	–	–	fpzip_24	2.68e-5	5.84e-7	.38	8.88e-15
PRECCDZM	speck_24	7.44e-9	1.61e-9	.77	fpzip_16	3.89e-3	4.16e-4	.24	5.29e-23
OMEGAT	speck_16	2.24e-8	3.11e-9	.51	fpzip_24	1.09e-5	2.04e-7	.52	0.0
FLNS	speck_12	1.38e-5	2.72e-6	.38	fpzip_24	2.81e-5	5.19e-6	.42	0.0
VQ	speck_16	3.50e-9	3.82e-10	.51	fpzip_24	9.53e-6	6.52e-7	.48	0.0
NUMLIQ	–	–	–	–	fpzip_32	0.0	0.0	.46	5.96e-8
WSUB	–	–	–	–	fpzip_32	0.0	0.0	.43	0.0

Tabla 7.7: Tabla comparativa de los ratios de compresión CR entre los compresores SPECK y fpzip [7]

Atendiendo al resultado del ratio de compresión (CR) del modelo sobre el conjunto de datos de la temperatura de la superficie, puede decirse que el modelo desarrollado se comporta de manera similar al compresor fpzip y mejor que otros compresores sobre los datos climáticos del estado del arte estudiado. Aún no siendo el análisis sobre el mismo conjunto de datos, el resultado de fpzip sobre los datos del conjunto de datos de la temperatura de la superficie (TS) del modelo CESM que se muestra en la tabla 2.4, denota un rendimiento similar al obtenido en el presente proyecto. A continuación se muestran los mapas de temperaturas de la superficie correspondientes a la muestra real, la predicción

del modelo y la reconstrucción total de los datos, tomando la muestra 75 del conjunto de evaluación del 2021.

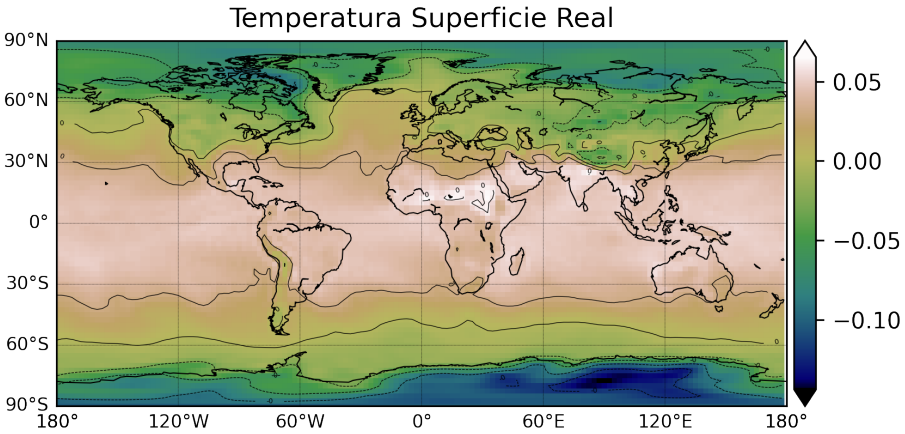

Figura 7.8: Mapa de la muestra real de la temperatura de la superficie

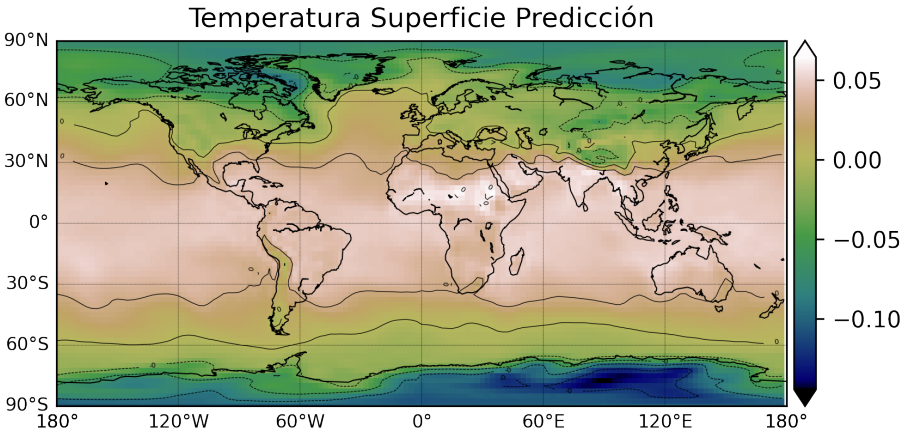

Figura 7.9: Mapa de la predicción del modelo de la temperatura de la superficie

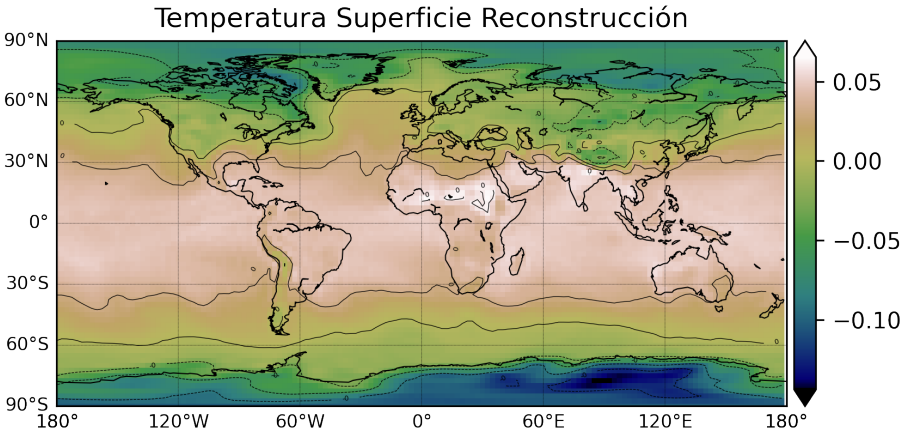

Figura 7.10: Mapa de la reconstrucción total de los datos de la temperatura de la superficie

7.2.2. Histograma de los residuos

Los datos reflejados en la figura 7.11 sobre la distribución de los residuos de las muestras originales y la predicción del conjunto de test de datos del 2021, muestran un mayor número de residuos concentrados en 0 que los resultantes en la figura 8.5 del conjunto de la temperatura en la tropopausa, correspondiente al mismo modelo y configuración.

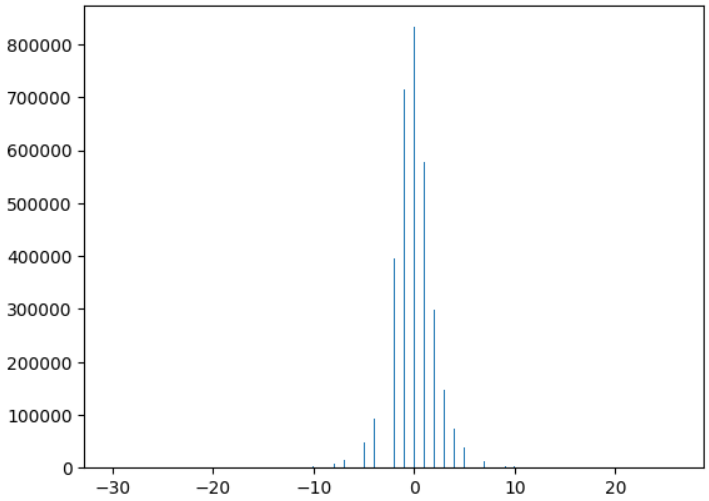

Figura 7.11: Histograma de los residuos Lookahead+Adabelief - RMSE

Capítulo 8

Conclusiones y líneas futuras

En este trabajo se ha evaluado un modelo para la compresión de datos de temperatura provenientes del NCEP/NCAR Reanalysis 1 utilizando un autoencoder convolucional junto con un procesado de los residuos y los datos comprimidos por el modelo. El proyecto se ha desarrollado siguiendo el enfoque de los trabajos de Baker et. al, en los cuales se describe la problemática que generan los datos científicos en cuanto al almacenamiento y cómputo de los mismos, así como se detalla una metodología para evaluar los distintos métodos de compresión de los datos que permita que éstos sean estadísticamente indistinguibles de la variabilidad natural del sistema climático. El modelo de autoencoder convolucional desarrollado durante el proyecto cumple con las métricas umbral descritas en los trabajos de Baker et. al en términos de coeficiente de determinación y presenta un ratio de compresión similar en rendimiento a los rangos del compresor fpzip sobre las variables climáticas de los mencionados trabajos, mejorando a otros compresores allí descritos.

Para futuros trabajos, puede ser interesante evaluar el modelo con distintas variables climáticas diferentes de la temperatura y que presenten una distribución espacial de los datos. También, atendiendo a un análisis más profundo de la naturaleza estadística de los datos, podría desarrollarse un modelo de compresión con técnicas de aprendizaje profundo más conveniente al tipo de variable que se estudie, como son los modelos generativos o los autoencoders variacionales (VAE). Otro apartado a considerar pudiera ser el uso de capas atencionales para transformadores en el contexto de las imágenes (ViT) [54]. También podrían explorarse otras técnicas de entrenamiento como son los optimizadores, por ejemplo Lion [55]. Finalmente, se podría evaluar el uso de otros modelos previamente entrenados para capturar características de las imágenes, como EfficientNet [56], y transferir el aprendizaje al modelo experimento.

Bibliografía

[1] Kevin Paul, Sheri Mickelson, John M. Dennis, Haiying Xu, and David Brown. Lightweight parallel python tools for earth system modeling workflows. pages 1985–1994, 2015.

[2] UCAR/Unidata. *Network Common Data Form (NetCDF)*. UCAR/Unidata, 2023. Version 4.8.1.

[3] Allison Baker, Dorit Hammerling, Sheri Mickelson, Haiying Xu, Martin Stolpe, Phillipe Naveau, Ben Sanderson, Imme Ebert-Uphoff, Savini Samarasinghe, Francesco De Simone, Francesco Carbone, Christian Gencarelli, John Dennis, Jennifer Kay, and Peter Lindstrom. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development*, pages 4381–4403, 2016.

[4] Allison H. Baker, Haiying Xu, John M. Dennis, Michael N. Levy, Doug Nychka, Sheri A. Mickelson, Jim Edwards, Mariana Vertenstein, and Al Wegener. A methodology for evaluating the impact of data compression on climate simulation data. In *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, page 203–214. Association for Computing Machinery, 2014.

[5] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

[6] Peter Lindstrom. Error distributions of lossy floating-point compressors. 10 2017.

[7] Allison Baker, Haiying Xu, Dorit Hammerling, Samuel Li, and John Clyne. Toward a multi-method approach: Lossy data compression for climate simulation data. pages 30–42, 2017.

[8] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

[9] N.M. Urban J.A. Saenz, N.Lubbers. Dimensionality-reduction of climate data using deep autoencoders. *arXiv preprint arXiv:1809.00027*, 2018.

[10] James Donnelly, Alireza Daneshkhah, and Soroush Abolfathi. Forecasting global climate drivers using gaussian processes and convolutional autoencoders. *Engineering Applications of Artificial Intelligence*, 128:107536, 2024.

[11] O. D. Levers, D. Herremans, A. Dipankar, and L. Blessing. Downscaling using deep convolutional autoencoders, a case study for south east asia. *EGUsphere [preprint]*, 2022.

[12] K.L.Polsterer S.Lerch. Convolutional autoencoders for spatially-informed ensemble post-processing. *arXiv preprint arXiv:2204.05102*, 2022.

[13] Doaa I. Ibrahim, Zhongwei Jiang, and Shaukat Ali Butt. An efficient compression of ecg signals using deep convolutional autoencoders. *Biomedical Signal Processing and Control*, 47:214–220, 2019.

[14] Shibani Santurkar, David Budden, and Nir Shavit. Generative compression. *arXiv preprint arXiv:1703.01467*, 2017.

[15] A.Cunningham F.Huszár L.Theis, W.Shi. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[17] François Chollet. *Deep Learning with Python*. Manning Publications, 2017.

[18] M. T. Hagan et al. *Neural Network Design*. Oklahoma State University, USA, 2nd edition, 2014.

[19] José Luis Sarmiento-Ramos. Aplicaciones de las redes neuronales y el deep learning a la ingeniería biomédica. *Revista UIS Ingenierías*, 19(4):1–18, 2020.

[20] Andrew Ng. Sparse autoencoder. CS294A Lecture notes.

[21] He Feng and Ren Teng. Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. 2019.

[22] Yasin and Abdulazeez. Image compression based on deep learning: A review. *AJRCOS*, 8(1):62–76, 2021. Article no.AJRCOS.67968.

[23] Visin Dumoulin. A guide to convolution arithmetic for deep learning. 2016.

[24] Rowel Atienza. *Advanced Deep Learning with TensorFlow 2 and Keras: Apply DL, GANs, VAEs, deep RL, unsupervised learning, object detection and segmentation, and more*. Packt Publishing Ltd, 2020.

[25] Song Kang and Sun. A deep similarity metric method based on incomplete data for traffic anomaly detection in iot. 2018.

[26] Roman Manakov, Manuel Rohm, and Volker Tresp. Walking the tightrope: An investigation of the convolutional autoencoder bottleneck. 2020.

[27] François Chollet. Building autoencoders in keras, 2016.

[28] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)*, 313:504–7, 08 2006.

[29] Kaggle. Kaggle: Your machine learning and data science community. `https://www.kaggle.com/`.

[30] Project Jupyter. Project jupyter. `https://jupyter.org/`.

[31] Python Software Foundation. Welcome to python.org. `https://www.python.org/`.

[32] Keras. Keras: Deep learning for humans. `https://keras.io/`.

[33] TensorFlow. Tensorflow: An open source machine learning framework for everyone. `https://www.tensorflow.org/`.

[34] Matplotlib Development Team. Matplotlib: Visualization with python. `https://matplotlib.org/`.

[35] NumPy Community. Numpy: The fundamental package for scientific computing with python. `https://numpy.org/`.

[36] Hossein Talebi and Peyman Milanfar. Learning to resize images for computer vision tasks. 2021.

[37] PixInsight. Interpolation algorithms. `https://pixinsight.com/doc/docs/InterpolationAlgorithms/InterpolationAlgorithms.html`.

[38] Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang. Image denoising using very deep fully convolutional encoder-decoder networks with symmetric skip connections. *CoRR*, abs/1603.09056, 2016.

[39] Hyeongyeom Ahn and Changhoon Yim. Convolutional neural networks using skip connections with layer groups for super-resolution image reconstruction based on deep learning. *Applied Sciences*, 10(6), 2020.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[41] Dan Hendycks and Kevin Gimpel. Gaussian error linear units (gelus). 2020.

[42] Varun Krishnaraj et al. Deep learning model for real-time image compression in internet of underwater things (iout). 2019.

[43] NOAA PSL. Ncep reanalysis daily tropopause data, 2024.

[44] NOAA PSL. Ncep reanalysis daily surface data, 2024.

[45] Eugenia Kalnay et al. The ncep/ncar 40-year reanalysis project. *Bull. Amer. Meteor. Soc.*, 77:437–470, 1996.

[46] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[47] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[48] Less Wright et al. Ranger21: a synergistic deep learning optimizer. *arXiv preprint arXiv:2106.13731*, 2021.

[49] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *arXiv preprint arXiv:2010.07468*, 2020.

[50] Leslie N Smith. Cyclical learning rates for training neural networks. *arXiv preprint arXiv:1506.01186*, 2015.

[51] Michael R Zhang, James Lucas, Jimmy Ba, and Geoffrey Hinton. Lookahead optimizer: k steps forward, 1 step back. *arXiv preprint arXiv:1907.08610*, 2019.

[52] Soonbin Lee, Jong-Beom Jeong, Inae Kim, and Eun-Seok Ryu. Learned image compression with frequency domain loss. pages 1–4, 2021.

[53] I.Frosio J.Kautz H.Zhao, O.Gallo. Loss functions for neural networks for image processing. *arXiv preprint arXiv:1512.03385*, 2017.

[54] Liwen Liu Hsin-Chien Liang Yi-Chi Wang Wan-Ling Tseng Chao Wang Che-Ta Chen Chia-Hao Chiang, Zheng-Han Huang and Ko-Chih Wang. Climate downscaling: A deep-learning based super-resolution model of precipitation data with attention block and skip connections. *arXiv preprint arXiv:2403.17847*, 2024.

[55] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms, 2023.

[56] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.