Aprendizaje máquina para la clasificación de imágenes geolocalizadas
Classification of geolocated images using Machine learning

Trabajo de Fin de Grado
Curso 2024–2025

Autor
Javier Saras González

Director
Rubén Fuentes Fernández

Calificación: *10*

Doble Grado en Ingeniería Informática y Matemáticas
Facultad de Informática
Universidad Complutense de Madrid

Aprendizaje máquina para la clasificación de imágenes geolocalizadas
Classification of geolocated images using Machine learning

Trabajo de Fin de Grado en Ingeniería Informática

Autor
Javier Saras González

Director
Rubén Fuentes Fernández

Convocatoria: *Junio 2025*

Calificación: ***10***

Doble Grado en Ingeniería Informática y Matemáticas
Facultad de Informática
Universidad Complutense de Madrid

13 de junio de 2025

Dedicatoria

Al niño curioso que llevo dentro, ese que se emociona adivinando en las películas los lugares donde se grabaron y que disfruta visitando todos esos rincones del mundo, aprendiendo sobre sus culturas y deleitándose con sus paisajes

Agradecimientos

Quiero expresar mi más sincero agradecimiento a mis padres, a mi hermana y al resto de mi familia, por su constante apoyo y por animarme a seguir adelante con este proyecto, incluso en los momentos en que parecía un reto demasiado grande para afrontarlo solo.

A Rubén, mi director de TFG, le agradezco profundamente la confianza que me brindó al permitirme desarrollar este tema con total libertad. Su orientación y apoyo en cada duda y dificultad que surgió a lo largo del proceso han sido fundamentales. Además, quiero agradecerle por haber sido mi profesor en la asignatura de Inteligencia Artificial, una disciplina fascinante que me abrió las puertas a un mundo de posibilidades y de conocimientos que quiero seguir explorando.

Resumen

Aprendizaje máquina para la clasificación de imágenes geolocalizadas

Determinar las coordenadas exactas de una imagen es una tarea visual sumamente compleja, resultando en un caso de estudio ideal para evaluar y desarrollar modelos de aprendizaje supervisado para tratar de resolver el problema. Para ello, se selecciona un amplio conjunto de datos etiquetado, equilibrado y representativo de la diversidad geográfica mundial; y se investigan maneras eficientes de gestionarlo y usarlo. El *dataset* se divide en entrenamiento, validación y test para, respectivamente, entrenar las redes neuronales, determinar cuándo parar dicho entrenamiento y evaluar el rendimiento final. Se propone una arquitectura que consta de un codificador de imágenes encargado de extraer características visuales relevantes, seguido de una red neuronal densa que estima la ubicación. Se exploran distintas configuraciones de parámetros y se definen varias funciones de pérdida para probar diferentes estrategias y ver cuál tiene un mejor desempeño, resultando en 36 modelos entrenados con la técnica *early stopping* para mitigar sobreajuste y subajuste. Después del entrenamiento, se evalúan los diferentes prototipos y se selecciona el que muestra un mejor desempeño en términos de consistencia en la precisión y en su capacidad de generalización ante nuevos datos, consiguiendo así un modelo competitivo con predicciones de menos de 900 km a más del 50 % del conjunto de test. Finalmente, se desarrolla una aplicación que integra ese modelo y permite estimar ubicaciones de nuevas imágenes. Todo el código asociado, los materiales adicionales y los modelos entrenados se pueden encontrar en https://drive.google.com/drive/folders/1YvVd-4UFtF6zzuGpNKgqjYX8kd002gRX?usp=sharing

Palabras clave

Geolocalización de Imágenes, Funciones de Pérdida de Geolocalización, Localizabilidad de Imágenes, CLIP, Perceptrón Multicapa

Abstract

Classification of geolocated images using Machine learning

Determining the precise coordinates of an image is a highly complex visual task, making it an excellent case study for evaluating and developing supervised learning models aimed at solving this problem. To achieve this, a large, geotagged, well-balanced, and representative dataset reflecting the world's geographic diversity is selected, and efficient methods to manage and use it are investigated. The dataset is divided into training, validation, and test sets, used respectively to train the neural networks, determine when to stop the training, and evaluate final performance. We propose an architecture composed of an image encoder responsible for extracting relevant features, followed by a dense neural network that estimates location. Different parameter configurations are explored, and several loss functions are defined to test different strategies and see which one performs better. This results in the training of 36 models, using the early stopping technique to mitigate overfitting and underfitting. After training, the different prototypes are evaluated, and the one showing the best performance in terms of accuracy and generalization to unseen data is selected, thus achieving a competitive model with prediction errors below 900 km in more than 50% of the test images. Finally, an application integrating the model is developed to allow location estimations of new images. All associated codes, additional materials, and trained models can be found at https://drive.google.com/drive/folders/1YvVd-4UFtF6zzuGpNKgqjYX8kd002gRX?usp=sharing

Keywords

Image Geolocation, Geolocation Loss Functions, Image Localizability, CLIP, Multilayer Perceptron

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

"Any sufficiently advanced technology is indistinguishable from magic"
— Arthur C. Clarke

1.1. Motivación

Determinar la ubicación precisa de una imagen teniendo en cuenta toda la superficie terrestre representa un desafío de alta complejidad en el campo de los algoritmos de visión computacional. Justamente por eso, resulta en un caso de estudio ideal para la evaluación y desarrollo de algoritmos de clasificación, en particular los basados en el aprendizaje automático como son las redes neuronales.

En este sentido, tratar este problema me proporciona una excelente oportunidad para poner en práctica todo lo aprendido en las asignaturas de Inteligencia Artificial (IA), especialmente los conocimientos sobre redes neuronales y clasificadores. Asimismo, ya sea realizando un máster u orientando mi futura trayectoria laboral, este proyecto se alinea con mis intereses futuros en lo relativo a la IA.

La aplicación de aprendizaje supervisado para la geolocalización de imágenes sigue siendo un área de investigación que tiene un amplio margen de exploración. Esto se debe a que hasta hace poco, la mayoría de los conjuntos de datos de imágenes geolocalizadas presentaban principalmente dos limitaciones significativas: (1) un fuerte sesgo geográfico debido a una sobre-representación de determinadas regiones del mundo y falta de diversidad, lo que hacía inabordable la geolocalización a escala global; y (2) un alto porcentaje de imágenes no localizables debido a su baja calidad o a la ausencia de elementos distintivos que permitan diferenciarlas geográficamente.

No obstante, recientemente se ha publicado `OpenStreetView-5M` (Astruc et al., 2024). Es un conjunto de datos de acceso abierto a gran escala que contiene más de 5,1 millones de imágenes *street view* geolocalizadas abarcando un total de 225 países y territorios. Este conjunto de datos mitiga en gran medida las limitaciones anteriormente mencionadas, lo que lo convierte en un recurso idóneo para el presente estudio.

Además, la geolocalización de imágenes presenta numerosas aplicaciones prácticas en diferentes campos. Entre ellos se encuentran la *navegación sin GPS*, útil en

situaciones donde necesitas saber tu ubicación exacta porque te has perdido en un lugar donde no hay cobertura; la *verificación de información periodística* mediante técnicas de *fact-checking* para validar si las imágenes fueron tomadas en el lugar que se afirma; y el apoyo en *investigaciones criminales* permitiendo rastrear la ubicación donde se hizo una fotografía, lo que puede contribuir a la identificación de sospechosos o víctimas.

1.2. Objetivos

El principal objetivo del trabajo es desarrollar un modelo de aprendizaje automático supervisado, lo más robusto posible, que pueda geolocalizar imágenes con bastante precisión y que sea capaz de adaptarse bien ante nuevos entornos geográficos. Para ello, se diseñarán una serie de *benchmarks* que permitirán evaluar y comparar distintas alternativas, identificando las mejores aproximaciones al problema.

No se busca un modelo que realice clasificaciones basadas en patrones superficiales o debidas a un sobreaprendizaje y memorización del conjunto de entrenamiento. En cambio, se pretende que aprenda a identificar características geográficas clave. Estas pueden incluir, por ejemplo, señales de tráfico, patrones arquitectónicos y elementos naturales como el clima o la vegetación.

Como se quiere un modelo que posea una comprensión más profunda y generalizable de los rasgos geográficos relevantes, resulta esencial disponer de un conjunto de entrenamiento que evite sesgos derivados de una sobre-representación de ciertas regiones terrestres. Ahí es donde entra, como ya se había mencionado, `OpenStreetView-5M`, un *dataset* que proporciona una base estandarizada y fiable. Este se empleará para poder entrenar y evaluar el modelo minimizando el riesgo de sesgo y favoreciendo la generalización a nuevas imágenes.

Por último, se busca desarrollar una aplicación que utilice el modelo previamente entrenado para realizar predicciones sobre imágenes. Esta aplicación permitirá al usuario cargar imágenes locales para así obtener unas coordenadas que estimen su ubicación geográfica. Asimismo, dispondrá de una interfaz sencilla para facilitar una interacción entre el modelo y el usuario.

1.3. Plan de Trabajo

A continuación, se presenta el plan de trabajo que se va a llevar a cabo para conseguir todos los objetivos anteriormente planteados.

1. *Selección y preparación del Conjunto de Datos.* Primero, se analizará y razonará la selección del *dataset* `OpenStreetView-5M`. Una vez se tenga seleccionado el conjunto de datos, se investigará una manera eficiente de almacenar y gestionar un *dataset* tan grande. Esta organización resultará imprescindible para acceder y procesar fácilmente los datos en las siguientes fases del proyecto. *(Duración estimada: 27/01/2025 - 09/02/2025, 2 semanas)*

2. *División del Conjunto de Datos.* El conjunto de datos se dividirá en tres partes (entrenamiento, validación y test). Esta separación es necesaria para evitar el *overfitting*, para determinar las mejores configuraciones de los parámetros y para asegurar al final una evaluación objetiva del rendimiento de los modelos. *(Duración estimada: 10/02/2025 - 16/02/2025, 1 semana)*

3. *Definición de los Modelos y sus Parámetros.* Se definirán las arquitecturas de los modelos que se van a utilizar junto con las opciones y los parámetros disponibles para desarrollarlos. De esta manera, se podrán experimentar distintas configuraciones y estrategias para buscar aquella que ofrezca un mejor rendimiento. *(Duración estimada: 17/02/2025 - 09/03/2025, 3 semanas)*

4. *Entrenamiento de los Modelos.* Una vez definidos los modelos y sus parámetros correspondientes, se procederá al entrenamiento de cada uno de ellos. Durante esta fase se ajustarán los hiperparámetros y se monitoreará todo el proceso de entrenamiento y validación. *(Duración estimada: 10/03/2025 - 13/04/2025, 5 semanas)*

5. *Evaluación del Rendimiento y Selección del Mejor Modelo.* Después del entrenamiento se evaluarán los diferentes modelos con el conjunto de test. Luego se analizarán las gráficas de evaluación junto con el resto de la información generada para comparar su rendimiento y tomar decisiones informadas. Basándose en estos resultados, se seleccionará el modelo con el mejor desempeño en términos de precisión y capacidad de generalización ante nuevos datos. Este será utilizado en la siguiente fase del proyecto. *(Duración estimada: 14/04/2025 - 27/04/2025, 2 semanas)*

6. *Desarrollo de la Aplicación.* Finalmente, se desarrollará la aplicación mencionada que dispondrá de una interfaz sencilla y fácil de usar para que el usuario pueda utilizar el modelo seleccionado en la fase anterior. Esta interfaz permitirá cargar nuevas imágenes y así obtener las coordenadas predichas por el modelo. *(Duración estimada: 28/04/2025 - 04/05/2025, 1 semana)*

Capítulo 2

Estado de la Cuestión

Como ya se ha mencionado en la introducción (ver Sección 1.1), la geolocalización de imágenes a escala planetaria sigue siendo un problema complejo debido a la inmensa diversidad de imágenes que pueden provenir de cualquier parte del mundo. Predecir la ubicación de una imagen con precisión no es una tarea trivial, ya que depende de múltiples factores como la calidad de la imagen (por ejemplo, que no tenga subexposición o sobreexposición debido a una falta o exceso de luz al tomar la fotografía, que no esté desenfocada o mal rotada), así como la presencia de detalles visuales suficientes como para ser localizada.

En particular, las imágenes tomadas en espacios cerrados, como por ejemplo una cocina o un salón, ofrecen menos pistas geográficas en comparación con aquellas capturadas en entornos exteriores, como las imágenes *street view*. Estas últimas permiten apoyarse en diferentes elementos geográficos para identificar su localización. Estos pueden ser, entre otros, la vegetación, el clima, el suelo, el tipo de arquitectura, los rasgos físicos de las personas (como el rostro, la complexión o el color de piel), la apariencia externa (estilo de vestir, peinados o accesorios), y los elementos culturales como el lenguaje o las señales de tráfico.

Este problema es particularmente relevante en la actualidad, más aún en una época donde cualquier dispositivo electrónico facilita la captura masiva de imágenes geolocalizables, permitiendo un mapeo prácticamente global. Esta gran fuente de información permite aplicar técnicas de aprendizaje supervisado, que necesitan de grandes cantidades de datos para entrenar modelos que sean capaces de estimar con precisión ubicaciones de imágenes.

A continuación, primero se van a analizar los enfoques actuales más relevantes en el campo de la geolocalización de imágenes, destacando sus características clave y evolución (Sección 2.1) y después se discutirán las principales limitaciones de dichos enfoques, así como su relación directa con la propuesta de este trabajo y los vacíos que motivan el planteamiento adoptado (Sección 2.2).

2.1. Principales Enfoques Actuales

Como se ha comentado ya, la geolocalización de imágenes es un problema abierto de gran interés y con aplicaciones relevantes en diferentes campos. Existen diferentes aproximaciones para resolverlo.

En primer lugar, cuando están disponibles, es posible resolverlo accediendo a los metadatos de las imágenes. Este es el caso reflejado en el artículo (Spahić, 2022) para el juego de `GeoGuessr` (GeoGuessr, 2025). Un juego donde los jugadores son soltados en alguna ubicación aleatoria de `Google Street View` (Google, 2024) y tienen que adivinar la localización usando solo las pistas geográficas visibles. En ese artículo se explica claramente cómo existen algunas herramientas para hacer trampas en el juego accediendo a los metadatos de las imágenes, permitiendo obtener su ubicación exacta en cuestión de milisegundos. Nuestro trabajo prescinde de estas propuestas puesto que no se basan en la información visual de la imagen.

Dejando de lado los métodos anteriores, en el ámbito académico se han desarrollado diferentes líneas de trabajo para abordar el problema a partir de enfoques de aprendizaje automático. Una de las aproximaciones pioneras es la basada en recuperación de imágenes, aplicada en modelos como `IM2GPS` (Hays y Efros, 2008) e `IM2GPS Deep Learning` (Vo et al., 2017). En este caso, se compara una imagen desconocida con una base de datos etiquetada para estimar su ubicación mediante la similitud con las imágenes más cercanas en un espacio de características visuales. En la versión original se sacan varios grupos de características equiparables, como puede ser una reducción de las imágenes a tamaños muy pequeños de 16×16 píxeles o histogramas de color en el espacio `CIELAB` (donde se cuentan los píxeles en función de su luminosidad (L), sus componentes verde-rojo (a) y azul-amarillo (b), véase la Figura 2.1 para entender mejor ese espacio tridimensional que muestra otra forma de describir colores). En versiones más modernas, como `IM2GPS Deep Learning`, se emplean redes neuronales convolucionales (CNNs) para extraer representaciones más profundas y robustas de los rasgos. Una vez teniendo las imágenes en ese espacio de características, se comparan con el conjunto de entrenamiento para ver las más parecidas y estimar su ubicación.

Otro enfoque más reciente es el desarrollado en `OpenStreetView-5M` (Astruc et al., 2024), donde se utiliza un codificador de imágenes, como `CLIP`, para traducir las fotografías a vectores de características, los cuales se introducen en un módulo de geolocalización implementado con una red neuronal, que mapea ese vector a unas coordenadas reales. Este módulo puede ser una regresión directa o un modelo híbrido de clasificación sobre una partición de la tierra en regiones. La división se realiza mediante un árbol `QuadTree` (Samet, 1984), que divide iterativamente el mundo en cuadrantes hasta alcanzar una profundidad de 10 o hasta que una hoja contenga menos de 1000 imágenes del conjunto de entrenamiento. De esta manera se llegan a desarrollar 11 000 regiones distintas en todo el globo terráqueo y el modelo devuelve la probabilidad de que la imagen pertenezca a cada una de esas regiones. Asimismo, tienen otro módulo que a partir de la imagen te devuelve las coordenadas que pronostica en cada una de esas regiones y por eso la predicción final se obtiene combinando la región más probable con la coordenada estimada dentro de ella.

Podemos identificar así tres tipos de enfoques principales. En primer lugar, los

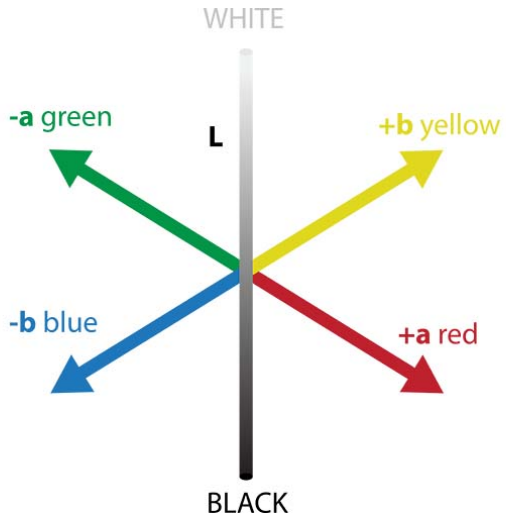

Figura 2.1: Espacio de colo tridimensional CIELAB (Datacolor, 2023)

métodos que explotan los metadatos de las imágenes, que no requieren de entrenamiento ni aprendizaje para funcionar, puesto que no se basan en técnicas de aprendizaje automático. En segundo lugar, los enfoques basados en recuperación de imágenes donde la predicción final depende de la similitud visual con ejemplos previos. Estos métodos realizan una búsqueda mejor fundamentada de la ubicación. Sin embargo, solo son efectivos cuando el conjunto de entrenamiento es extenso y bien distribuido, ya que tienen una fuerte dependencia con la base de datos de la que hacen la regresión y acaban resultando ineficaces ante imágenes muy distintas a las previamente vistas debido a su naturaleza basada en reconocimiento. Por último, el enfoque basado en aprendizaje profundo que deja a un lado el reconocimiento literal de lugares para utilizar una red neuronal y llevar a cabo un mapeo de características a ubicaciones reales. De esta manera, busca aprender a identificar patrones generales para ser más robusto frente a datos nuevos y diversos y dejar de depender exclusivamente de la existencia de un ejemplo idéntico en el conjunto de entrenamiento.

2.2. Limitaciones y Relación con este Trabajo

A pesar de los avances recientes, la geolocalización automática de imágenes sigue enfrentando varias limitaciones importantes. Una de las más destacadas es la falta de conjuntos de datos de alta calidad: se requieren *datasets* que posean imágenes bien etiquetadas y con una resolución razonable y que además tengan características suficientes como para permitir su localización. No todas las imágenes son igual de útiles en este contexto: una fotografía de una mascota aporta mucha menos información geográfica que una imagen al aire libre donde se pueda ver el clima, la vegetación y algunos elementos culturales como el idioma o las señales de tráfico.

Además, la mayoría de los conjuntos de datos disponibles suelen estar fuertemente sesgados hacia regiones urbanas o países occidentales, dejando muchas áreas del planeta subrepresentadas. Esta carencia dificulta la generalización de los modelos

a una escala global. Y aunque internet constituye una fuente masiva de datos que podría resolver parte del problema, el acceso restringido y costoso a muchos de estos conjuntos dificulta significativamente el desarrollo de soluciones competitivas.

Estas limitaciones afectan tanto a los enfoques basados en recuperación como a los modelos que intentan aprender patrones geográficos más generales. No obstante, el conjunto `OpenStreetView-5M` contribuye a resolver parcialmente esas limitaciones al ofrecer un dataset más extenso y diverso.

En este trabajo se plantea una solución basada en aprendizaje supervisado que busca reducir la dependencia en el *dataset* del reconocimiento directo de lugares y, en su lugar, fomentar el aprendizaje de patrones geográficos generales a partir de las características visuales de las imágenes. Para ello, se empleará un codificador de imágenes que proyecte las imágenes a un espacio de representación y, posteriormente, se aplicará un modelo de regresión de tipo red neuronal que estime las coordenadas geográficas a partir de dicha representación.

La propuesta comparte similitudes con el trabajo `OpenStreetView-5M` (Astruc et al., 2024), pero introduce varios cambios. En primer lugar, se adoptan técnicas para mitigar el sobreajuste, mediante una partición del conjunto de entrenamiento en uno de validación, con el objetivo de obtener modelos más generalizables. En segundo lugar, se experimentará con diferentes arquitecturas de codificadores de imágenes y distintas configuraciones de la red neuronal para evaluar su impacto en el rendimiento.

Finalmente, se abordará un aspecto poco tratado en trabajos anteriores: el diseño y estudio de distintas funciones de pérdida que intentan discutir nuevas alternativas de penalización en el aprendizaje con redes neuronales. De esta manera, se pueden comparar diferentes estrategias que permitan priorizar determinados aspectos en la predicción (como precisión local, consistencia global o robustez frente a ruido), en lugar de centrarse únicamente en minimizar la distancia geodésica entre la ubicación predicha y la real. Se planteará así una evaluación diferente, más exhaustiva, que no se limite simplemente a promedios de error, sino que busque modelos con un buen equilibrio entre precisiones muy precisas y consistentes, consiguiendo así un modelo más robusto para el problema de geolocalización de imágenes.

Capítulo 3

Marco Teórico

En este capítulo se presenta la base conceptual y el contexto científico y técnico que sustenta este trabajo, facilitando así un mejor entendimiento del desarrollo y de los resultados obtenidos. El contenido se estructura en varios apartados.

Primero, se comenta sobre el perceptrón multicapa ya que, dentro del aprendizaje automático supervisado, es una de las aproximaciones populares en el estado del arte para la geolocalización de imágenes.

Después, se presenta el conjunto de datos y cómo en *Machine Learning* (ML) se suele dividir en diferentes particiones según cómo se quiera actuar.

A continuación, se explica el problema de sobreajuste y subajuste en el aprendizaje supervisado y cómo estos pueden llegar a afectar al rendimiento y la capacidad de generalización del modelo ante nuevos datos.

Seguidamente se discute sobre el proceso de entrenamiento de las redes neuronales y la importancia del *Early Stopping* para evitar el sobreajuste. Además, se detallan los parámetros necesarios para configurar el entrenamiento, junto con los optimizadores para guiarlo y las funciones de pérdida para medir su desempeño.

Finalmente, se analiza la importancia del preprocesamiento de los datos y se describen diferentes enfoques que permiten adaptar los datos en bruto para que sean más adecuados y útiles para el aprendizaje.

3.1. Perceptrón Multicapa

Como se expone en (Popescu et al., 2009), el Perceptrón Multicapa, comúnmente conocido como `MLP` (*Multilayer Perceptron*, por sus siglas en inglés), es una de las arquitecturas más conocidas y ampliamente utilizadas dentro del campo de las redes neuronales. Esta red se compone de múltiples capas, incluyendo al menos tres niveles fundamentales: (i) una capa de entrada, con tantas neuronas como variables de entrada tenga el modelo; (ii) una o varias capas ocultas, cada una con un número variable de neuronas que incorporan funciones de activación; y (iii) una capa de salida, compuesta por tantas neuronas como número de variables de salida y que también utilizan funciones de activación. La incorporación de capas ocultas intermedias tiene como propósito permitir que el modelo genere representaciones progresivamente más

abstractas de los datos.

Cada neurona en estas capas puede entenderse como un perceptrón simple (véase la Figura 3.1), es decir, una unidad de procesamiento que realiza una combinación lineal de sus m variables de entrada, asignando a cada una de ellas un peso, y sumando un valor adicional denominado *bias* (sesgo). Esta combinación le permite al perceptrón simple "configurar" un hiperplano en el espacio de entrada separando dos clases que sean linealmente separables, es decir, que puedan situarse a cada lado del hiperplano y potencialmente usar dicho perceptrón para clasificar esas clases. Sin embargo, esta capacidad de separación lineal presenta limitaciones en problemas más complejos, como el caso clásico del operador lógico `XOR`, en el que las clases no pueden separarse mediante una única frontera lineal sino que necesitan dos rectas para diferenciarlas (véase la Figura 3.2). Para superar esta restricción, se introducen las funciones de activación para aportar no linealidad. Sin estas funciones, cada capa realizaría únicamente una combinación lineal de las salidas de la capa anterior, y como una combinación lineal de combinaciones lineales sigue siendo lineal, el modelo no podría representar relaciones no lineales, limitando gravemente su capacidad de generalización.

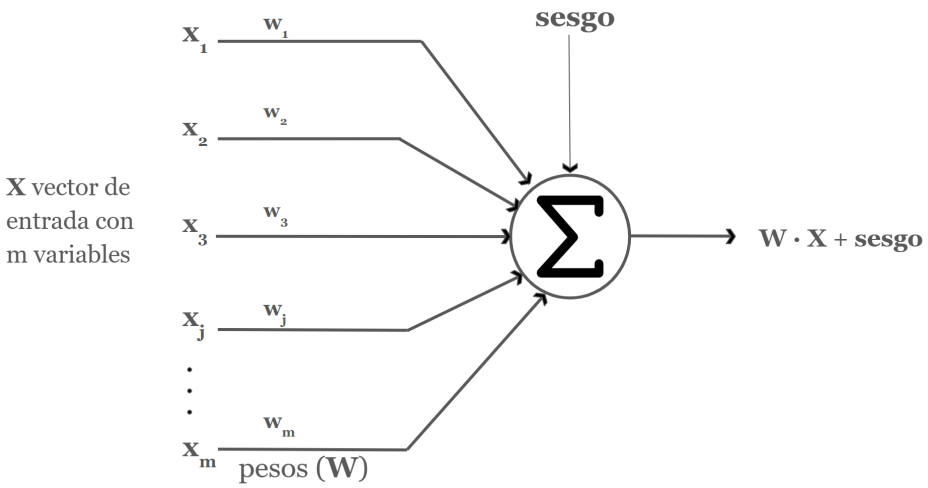

Figura 3.1: Perceptrón Simple

Esta arquitectura sigue un enfoque *feed-forward*, lo que implica que las señales se transmiten unidireccionalmente a través de las capas, desde la entrada hasta la salida, sin bucles ni retroalimentación. Además, otra de sus características fundamentales es que sus capas son *densamente conectadas* (*fully connected*, en inglés), lo que significa que cada neurona de una capa se conecta con todas las neuronas de la capa anterior y de la capa siguiente. Esta conectividad completa es precisamente la que facilita la capacidad del modelo para aprender patrones sofisticados y representar funciones de alta complejidad.

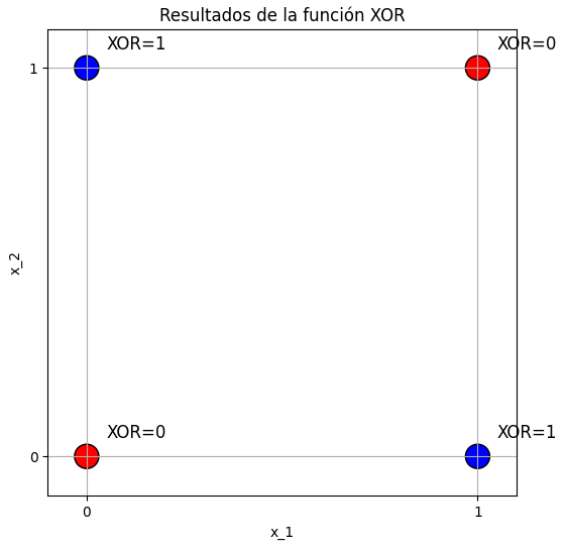

Figura 3.2: Limitación XOR

3.1.1. Funciones de Activación

Las funciones de activación resultan esenciales para superar la limitación de la linealidad y así poder representar relaciones complejas no lineales. Estas funciones se aplican a las salidas de las neuronas entre capas intermedias, así como en la capa de salida. La capa de entrada no suele incorporar una función de activación, permitiendo que los datos originales entren en crudo al modelo y sean los pesos de la primera capa los encargados de proporcionarles inicialmente una subrepresentación. Además, el sesgo desempeña un papel clave al permitir ajustar la región en la que se activa la función de activación, aumentando así la flexibilidad de la neurona.

Una propiedad clave que deben cumplir las funciones de activación es la derivabilidad (o al menos ser casi derivables) para poder calcular los gradientes durante el proceso de retropropagación (*backpropagation*, en inglés). Este proceso es fundamental en el entrenamiento de redes neuronales, ya que permite ajustar los pesos mediante el descenso de gradiente con el objetivo de minimizar la función de pérdida, y con ello mejorar la relación entre las variables de entrada y las de salida. Todos los conceptos relativos a la función de pérdida y cómo funciona toda esa actualización de pesos se explicará mejor en la Sección 3.4.3.

A continuación, se presentan algunas de las funciones de activación más comunes. Las expresiones matemáticas y representaciones gráficas utilizadas (ver Figura 3.3) se han extraído de (PyTorch, 2025):

a) *Función Identidad.* Función lineal utilizada habitualmente en la capa de salida para problemas de regresión no acotados. Su forma trivial facilita interpretaciones directas.

b) *Función Logística (Sigmoide).* Función sigmoidal que toma valores en el intervalo $(0, 1)$. Es acotada, monótona creciente y diferenciable. Su fórmula es: $\sigma(x) = \frac{1}{1+e^{-x}}$.

c) *Tangente Hiperbólica.* También sigmoidal, pero con rango en el intervalo $(-1, 1)$. Comparte con la función logística las propiedades de ser acotada, monótona creciente y diferenciable. Su fórmula es: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

d) *ReLU (Rectified Linear Unit).* Función por tramos: uno lineal para valores positivos y otro constante (cero) para los negativos. Aunque no es derivable en el punto $x = 0$, el de unión de los tramos, en la práctica se le asigna un valor convencional para facilitar el cálculo del gradiente (por ejemplo en `PyTorch` (PyTorch, 2025) donde ese valor es cero). Esta función es ampliamente utilizada en *Deep Learning* debido a varias razones: su alta linealidad en los subtramos reduce mucho el consumo computacional; su naturaleza no acotada evita la saturación por el extremo positivo pudiendo ser ventajoso en la fase de aprendizaje; y a diferencia de las anteriores, su capacidad de inhibir la activación de algunas neuronas al permitir salidas completamente nulas (las otras se aproximan infinitamente) resulta beneficiosa puesto que puede llegar a reducir tiempos de cálculo en redes muy profundas. Su fórmula es: $\text{ReLU}(x) = \text{máx}(0, x)$.

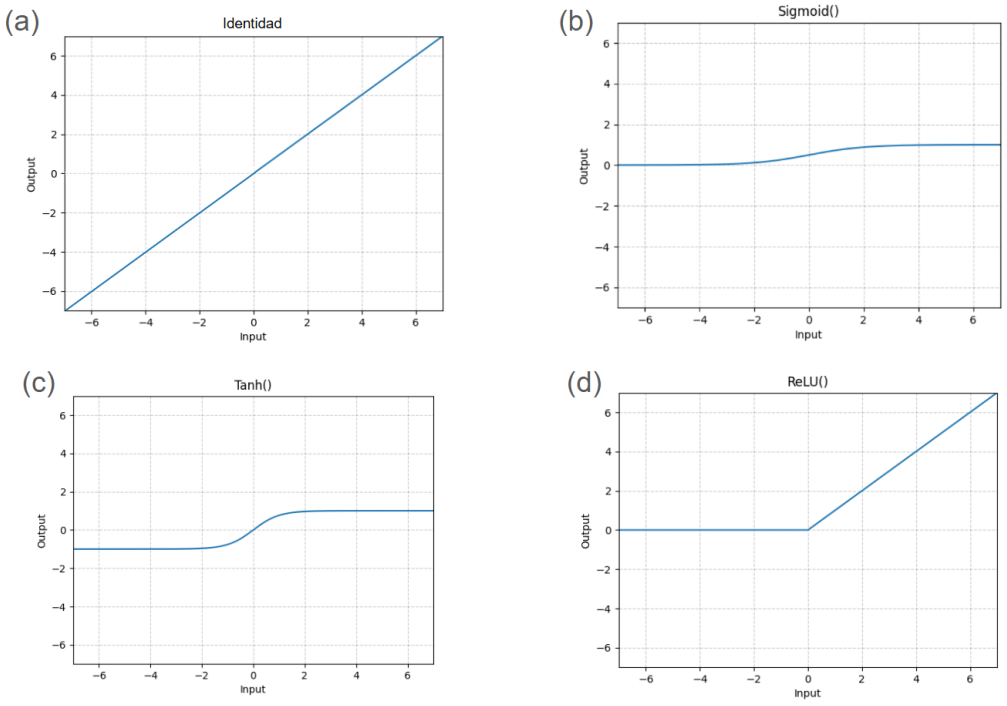

Figura 3.3: Ejemplos de Funciones de Activación (PyTorch, 2025)

3.2. Conjuntos de Datos

En *Machine Learning* es habitual que el modelo requiera ser ajustado o entrenado antes de ser usado. Para llevar a cabo estos retoques, se utiliza una parte del conjunto de datos disponibles conocida como conjunto de entrenamiento. Después,

tratando de ver cuán bueno es nuestro modelo ante datos desconocidos, se recurre a un conjunto de prueba independiente (también llamado conjunto de test). Esto acaba dando lugar a una partición simple del *dataset* en dos subconjuntos complementarios: entrenamiento y test.

La partición anterior puede resultar insuficiente: ¿cómo podemos parar el entrenamiento del modelo para evitar que este, por un lado, sobreaprenda del conjunto de entrenamiento, pero, por el otro, no se quede corto en ese aprendizaje?

Para ello, se requiere de un conjunto de datos extra que no se utilice para entrenar, pero sí permita decidir la mejor configuración del modelo, por ejemplo, ayudando a determinar aspectos como el número óptimo de épocas. Este rol lo cumple el conjunto de validación, ya que el conjunto de test debe reservarse exclusivamente para la evaluación final del modelo. Es fundamental que el modelo no tenga acceso al conjunto de test en ninguna fase del entrenamiento, previa a la fase de test final, asegurando así una evaluación objetiva y sin sesgos.

3.3. Overfitting vs Underfitting

El aprendizaje supervisado se basa en construir un modelo utilizando un conjunto de datos etiquetados para entrenarlo. Si el modelo se entrena demasiado tiempo, puede aprender no solo los patrones relevantes sino también el "ruido" o la información superflua de los datos de entrenamiento. Esto hace que el modelo se especialice demasiado en sus detalles y patrones específicos y se vuelva incapaz de extrapolar a otros datos no vistos. Este fenómeno se llama sobreajuste (*overfitting*, en inglés) (Ying, 2019). En contraste, el subajuste (*underfitting*, en inglés) ocurre cuando el modelo no se entrena el tiempo suficiente y no logra identificar patrones importantes durante el entrenamiento que capturen una relación significativa entre las variables de entrada y de salida. Acaba en su lugar aprendiendo información vaga e insuficiente, resultando en un rendimiento deficiente tanto con el conjunto de entrenamiento como con datos nuevos.

Para introducir el concepto del compromiso en complejidad (*trade-off in complexity*, en inglés) tengo que definir primero los siguientes conceptos (GeeksforGeeks, 2025):

- El sesgo (*bias*, en inglés) es el error sistemático en el que se puede incurrir cuando el modelo es muy simple, no aprende suficientes detalles de los datos y hace suposiciones incorrectas.

- La varianza (*variance*, en inglés) es la sensibilidad en las variaciones de las predicciones del modelo cuando este aprende demasiado de los datos de entrenamiento.

- La tasa de error es una proporción entre las predicciones incorrectas del modelo con respecto al total de predicciones.

El *overfitting* se caracteriza por tasas de error bajas en el conjunto de entrenamiento y una alta varianza. Esto se debe a que, al haber aprendido en exceso, el

modelo se ha adaptado demasiado a los detalles particulares del conjunto de entrenamiento, reduciendo su sesgo, pero aumentando su varianza. Otro indicador adicional es cuando los datos de entrenamiento presentan una baja tasa de error, mientras que los datos de test una muy alta.

Por el otro lado, los modelos subajustados presentan un alto sesgo, lo que se traduce en una alta tasa de error en entrenamiento y una menor varianza en sus predicciones. Esto ocurre porque el modelo no ha aprendido lo suficiente, lo que resulta en que ni siquiera sepa predecir bien los datos de entrenamiento. Parece que concluye los resultados aleatoriamente, provocando así que al final su desempeño sea mediocre, mostrando una baja variabilidad en sus resultados lo que reduce la varianza.

A medida que el modelo avanza en su proceso de entrenamiento, la varianza aumenta y el sesgo disminuye, pasando así de un modelo subajustado a un modelo sobreajustado. Esto ilustra ese *bias-variance trade-off in complexity*, un concepto clave en el aprendizaje automático que describe el equilibrio entre la varianza y el sesgo (véase la Figura 3.4). Este equilibrio implica un balance entre precisión y consistencia: la precisión en predecir correctamente las salidas a partir de los datos de entrada, y la consistencia en mantener un comportamiento del modelo estable y sólido frente a datos nuevos. Por esa razón, al entrenar un modelo usando el aprendizaje supervisado, se busca alcanzar un punto óptimo o de equilibrio (*sweet spot*, en inglés) entre el *overfitting* y el *underfitting*. Este punto garantiza que el modelo no haya aprendido en exceso ni en defecto y que sea capaz de generalizar adecuadamente a datos no vistos durante el entrenamiento.

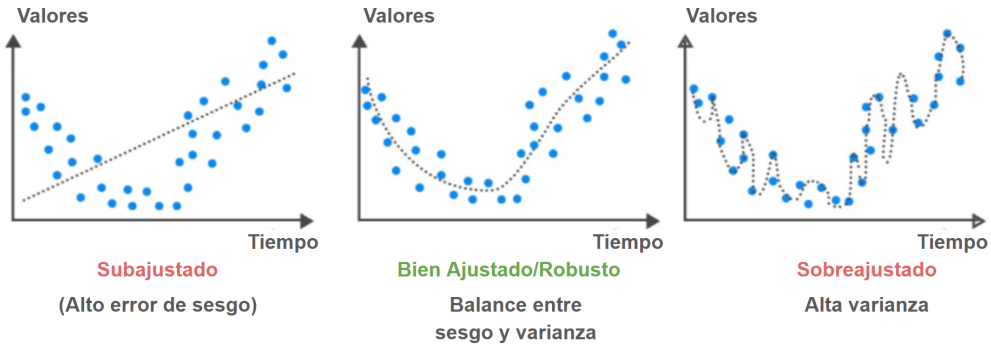

Figura 3.4: Overfitting vs Underfitting (sesgo-varianza) (Singh, 2021)

3.4. Entrenamiento del Modelo

En las redes neuronales, el entrenamiento se realiza a lo largo de varias épocas, durante las cuales el modelo ajusta sus configuraciones iterativamente para encontrar el mejor conjunto de parámetros que minimice la función de pérdida. Una época es una pasada completa o parcial del conjunto de datos de entrenamiento (la distinción entre completa o parcial se aclara en la Sección 3.4.2.3). Y es que durante cada época, el modelo va procesando ejemplos de entrenamiento para actualizar sus pesos en

función del error cometido y así mejorar la precisión.

3.4.1. Early Stopping

A medida que el entrenamiento avanza, los pesos pueden volverse excesivamente específicos para reducir el error del conjunto de entrenamiento, aumentando el riesgo de sobreajuste. El modelo podría empezar a memorizar patrones demasiado particulares en lugar de aprender representaciones generales que se transfieran eficazmente a nuevos datos.

El *early stopping* (Ying, 2019) es una técnica ampliamente utilizada en el entrenamiento de redes neuronales para prevenir ese sobreajuste. Esta técnica consiste en monitorizar el rendimiento del modelo en un conjunto de validación y detener el proceso de entrenamiento cuando dicho rendimiento deje de mejorar. De este modo, se evita que el modelo se adapte en exceso a los datos de entrenamiento y se favorece su capacidad de generalización ante datos no vistos. Esta estrategia permite optimizar el equilibrio entre sesgo y varianza, mejorando la robustez del modelo en escenarios reales.

Como se puede ver en la Figura 3.5, donde el eje de abscisas representa las épocas de entrenamiento de la red neuronal y el eje de ordenadas el error (ε), la línea azul indica el error del conjunto de entrenamiento y la línea roja el de validación. Si el modelo continúa aprendiendo después del punto señalado, el error de validación comenzaría a estancarse o incrementarse mientras que el de entrenamiento continuaría decreciendo.

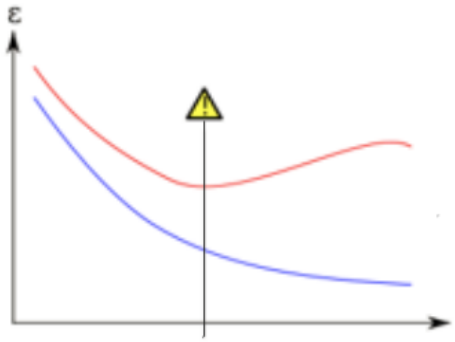

Figura 3.5: Early Stopping (Ying, 2019)

Si paramos el entrenamiento demasiado pronto (antes del punto marcado), el modelo sufrirá subajuste, mientras que si lo prolongamos en exceso (después del punto), sufrirá sobreajuste. Por ello, el objetivo es encontrar el punto óptimo donde parar el entrenamiento, el *sweet spot*. Solo así conseguiremos un equilibrio entre *overfitting* y *underfitting*.

No sería prudente detener el entrenamiento inmediatamente después de una única época en la que el error del conjunto de validación deje de mejorar. Por esta razón, en la práctica se introduce el parámetro de paciencia, que define el número de épocas adicionales antes de detenerse que se permite al modelo seguir entrenando sin

mejoras en la métrica de validación. De esta manera, el modelo continúa entrenándose mientras que, en cada época, se calcula la media de la pérdida en el conjunto de validación. Solo cuando esta pérdida se haya mantenido sin mejorar durante un número de épocas igual al valor de paciencia, se decide interrumpir el entrenamiento. Además, para garantizar que se alcanza el punto de equilibrio observado en la imagen, el modelo no se detendrá simplemente en la última época entrenada. En su lugar, se restaurará al estado en el que logró su mejor desempeño en validación. Esto es crucial, ya que seleccionar el último modelo entrenado podría implicar elegir uno potencialmente sobreajustado.

Digo potencialmente sobreajustado porque existe una consideración a tener en cuenta. El valor de paciencia es esencial, ya que un valor demasiado bajo podría detener el entrenamiento del modelo prematuramente sin haber tenido suficiente tiempo para aprender patrones importantes. Por el contrario, un valor excesivamente alto podría no ser eficaz para evitar el sobreajuste. Esto último se debe a que el conjunto de validación no siempre refleja con total fidelidad un conjunto de datos completamente nuevos. Es posible que la métrica de validación se mejore sin que ello implique una mejor capacidad de generalización. En otras palabras, el modelo podría sobreajustarse al conjunto de validación, de forma análoga a como ocurre con el conjunto de entrenamiento cuando se entrena durante demasiado tiempo. Este fenómeno puede darse cuando una configuración específica, que ya se ha ajustado en exceso a los datos de entrenamiento, también se adapta inusualmente bien al conjunto de validación, reduciendo su pérdida de manera engañosa. Sin embargo, esta mejora podría no ser genuina, sino el resultado de haber incorporado indirectamente información espuria del conjunto de validación en la red neuronal, lo que afectaría su rendimiento real en datos completamente nuevos. Aunque este efecto no es común, un valor de paciencia excesivamente alto aumentaría las probabilidades de que ocurra.

3.4.2. Parámetros de Entrenamiento

En esta sección, se discutirán los hiperparámetros que se configuran antes de comenzar el entrenamiento, los cuales son fundamentales para la dinámica de entrenamiento, la utilización de recursos, el tiempo de entrenamiento y el rendimiento final del modelo.

3.4.2.1. Paciencia

Como se introdujo en la Sección 3.4.1, este parámetro determina cuántas épocas pueden transcurrir sin que la pérdida del conjunto de validación mejore antes de que el entrenamiento se detenga. Es esencial para evitar el sobreajuste. Además, gracias a este valor y al *early stopping*, determinamos indirectamente el número total de épocas durante las cuales la red será entrenada.

3.4.2.2. Tasa de Aprendizaje

La tasa de aprendizaje (Interactive Chaos, 2025) influye directamente en cómo se optimizan los pesos de la red a través del descenso por gradiente. En términos

generales, la tasa de aprendizaje controla la magnitud de los pasos dados para ajustar los pesos durante cada iteración del entrenamiento. Intuitivamente, si es demasiado alta, el modelo puede "saltar" el mínimo de la función de pérdida, mientras que si es demasiado baja, el modelo puede tardar demasiado en converger o ser más propenso a atascarse en mínimos locales. Aunque el optimizador (que se discutirá más adelante) define el algoritmo para actualizar los pesos basado en alguna variante del descenso de gradiente, es la tasa de aprendizaje la que controla el tamaño de las actualizaciones.

Estas limitaciones se podrían resolver con una variación dinámica de la tasa de aprendizaje que comenzase con grandes saltos para explorar muchas regiones del espacio y así encontrar la zona de configuración de pesos más prometedora, evitando un estancamiento inicial en un mínimo local. Una vez esté localizada dicha región, se podría ir reduciendo la tasa para realizar ajustes más finos y precisos para así encontrar el óptimo global sin desviarse demasiado.

3.4.2.3. Batch Size

El *batch size* (Ultralytics, 2025) se refiere a la cantidad de muestras que el modelo procesa antes de actualizar los pesos durante una iteración del entrenamiento (época). Dado que procesar todo el conjunto de datos a la vez es muy costoso computacionalmente, se hace más viable dividir los datos en lotes más pequeños y manejables. De este modo, los pesos del modelo se actualizan después de procesar cada lote, haciendo que el entrenamiento sea más eficaz. La elección de este hiperparámetro es crucial, ya que afecta significativamente a la dinámica del entrenamiento, a la utilización de los recursos y al rendimiento final:

- *Velocidad de entrenamiento*: Por un lado, cuanto mayor sea el tamaño del lote, si se puede paralelizar, se conducen a épocas de entrenamiento más rápidas, ya que permiten una mejor utilización de las capacidades de procesamiento paralelo del *hardware*, como las GPU (se procesan más datos por ciclo de cálculo). Sin embargo, si no se puede paralelizar, tener lotes más pequeños acelera el entrenamiento al considerar menos datos en cada iteración.

- *Consumo de memoria*: A mayor tamaño del lote, mayor es el consumo de memoria. El lote debe caber en la memoria disponible del *hardware* (por ejemplo, en la RAM de la GPU). Superar este límite puede generar errores (como ocurre con las GPU y PyTorch) o ralentizar drásticamente el entrenamiento, ya que los datos tendrán que rotarse entre la memoria y el disco (como ocurre con la RAM de la CPU y el *swap* en disco).

- *Rendimiento y generalización del modelo*: Los lotes más pequeños provocan más actualizaciones de los pesos, lo que introduce más ruido en la estimación del gradiente ya este se basa en una cantidad menor de datos. Este ruido puede actuar como una forma de regularización, ayudando a evitar que el modelo se ajuste demasiado a los datos específicos del entrenamiento protegiendo de *overfitting* y favoreciendo una mejor generalización. Además, el ruido podría ayudar a escapar de mínimos locales, ya que otro lote podría no estar atrapado en el mismo mínimo. Sin embargo, es un arma de doble filo, puesto que

lotes excesivamente pequeños pueden hacer que el entrenamiento sea inestable afectando negativamente a la convergencia y calidad del modelo final. Por el contrario, los lotes más grandes dan lugar a estimaciones más precisas y estables del gradiente. Esto ayuda a que la red converja de manera más fluida y con menos oscilaciones en el proceso de optimización. Sin embargo, también aumenta el riesgo de sobreajuste. Los lotes grandes pueden hacer que el modelo converja rápidamente a mínimos locales que no son necesariamente los mejores, y como el modelo está viendo muchas muestras a la vez, puede aprender patrones específicos de los datos de entrenamiento, lo que aumenta la probabilidad de que se "memorizen" detalles innecesarios en lugar de aprender patrones generales.

Existen tres alternativas para fijar el tamaño del lote: (i) *Batch Gradient Descent* cuando el lote es todo el conjunto de datos; (ii) *Stochastic Gradient Descent (`SGD`)* cuando el lote es un solo elemento; y (iii) *Mini-Batch Gradient Descent*, que es una solución intermedia donde el lote contiene $n > 1$ muestras.

3.4.3. Funciones de Pérdida y Métricas

En el contexto de las redes neuronales, la función de pérdida o función de error (Bergmann y Stryker, 2025) es un componente fundamental para evaluar el rendimiento del modelo, ya que cuantifica la discrepancia entre las predicciones del modelo y los valores reales esperados. La optimización del modelo se centra en esta función, dado que el proceso de entrenamiento consiste en ajustar los parámetros del modelo para minimizar dicha pérdida.

Como se explicó en la Sección 3.4.2.3, durante el entrenamiento, el modelo realiza predicciones sobre un lote (*batch*) de datos tomados del conjunto de entrenamiento. A partir de estas predicciones, se calcula el error promedio entre los valores predichos y los reales, lo que permite retroalimentar el sistema y ajustar los pesos del modelo.

Cabe destacar que las funciones de pérdida no se limitan a ser simples métricas de evaluación. Su propósito va más allá de medir el éxito del modelo; también sirve como entrada para algoritmos de optimización que ajustan los parámetros del modelo con el fin de minimizar esta pérdida. Algoritmos de optimización (optimizadores, véase Sección 3.4.4) como el descenso de gradiente y sus variantes se apoyan en el cálculo del gradiente de la función de pérdida, es decir, su derivada. Por ello, es indispensable que la función de pérdida sea diferenciable en todos sus puntos. A partir del gradiente, los optimizadores determinan la dirección y magnitud adecuadas para actualizar los parámetros del modelo y así reducir progresivamente la pérdida.

Las redes neuronales emplean habitualmente el mecanismo de retropropagación para calcular el gradiente de la función de pérdida tras un pase hacia adelante. Este proceso se inicia con la predicción sobre un lote de datos, a partir de la cual se calcula la pérdida. Luego, mediante reglas de la derivación en cadena, se realiza un pase hacia atrás (retropropagación) desde la capa de salida hasta la de entrada, determinando como cada peso y sesgo ha contribuido a la pérdida total. Con esta información, se actualizan los parámetros mediante derivadas parciales, repitiendo el proceso hasta alcanzar una pérdida suficientemente baja.

Las funciones de pérdida pueden agruparse en dos categorías principales: aquellas diseñadas para problemas de *regresión*, que cuantifican errores sobre variables continuas, y las orientadas a problemas de *clasificación*, que miden errores en variables discretas o categóricas. La elección entre una u otra depende directamente de la naturaleza del problema que se desea resolver. En tareas de regresión, la variable objetivo adopta valores numéricos continuos, mientras que en clasificación se trabaja con etiquetas que representan categorías distintas.

La elección de la función de pérdida es crucial, ya que diferentes funciones priorizan distintos tipos de error. Algunas penalizan con mayor severidad los valores atípicos, mientras que otras se enfocan en mejorar la precisión en casos cercanos al valor objetivo. Además, la complejidad computacional también puede ser un factor determinante, pues funciones más precisas en la penalización pueden requerir más recursos y tiempo de cómputo.

3.4.4. Optimizador

En el contexto de una `MLP`, un optimizador, según (PyTorch, 2025), es un objeto que implementa un algoritmo que decide cómo se actualizarán los parámetros de la red neuronal a partir de los gradientes calculados durante el proceso de retropropagación. Es decir, define las reglas que determinan cómo ajustar los pesos del modelo. Además de esta tarea principal, el optimizador gestiona el estado interno del proceso de optimización. Este estado incluye información relevante como el historial de gradientes, sus promedios y otros parámetros auxiliares que facilitan decisiones más informadas al actualizar los pesos, permitiendo ajustes más eficientes durante el entrenamiento.

3.5. Preprocesamiento de Datos

El preprocesamiento de datos (García et al., 2015) es el proceso de acondicionamiento de los datos brutos para que sean más adecuados y útiles para el aprendizaje automático. Esto puede implicar operaciones como: (i) limpiar datos para reducir el ruido, los errores y las inconsistencias; (ii) integrar datos combinándolos desde diferentes fuentes; (iii) transformar datos para tener un formato más apropiado para las siguientes tareas en las que se van a utilizar; y (iv) reducir datos, que incluye la selección y extracción de características o ejemplos relevantes para disminuir la dimensionalidad y el volumen sin perder información significativa.

A continuación, se van a describir diferentes ejemplos de tratamiento de datos.

3.5.1. Normalización

La normalización (García et al., 2015) es un tipo de preparación de datos donde se convierten los datos crudos y a priori no suficientemente buenos en datos correctos para ser usados en el algoritmo de aprendizaje automático concreto. Su objetivo principal es expresar todas las variables bajo la misma escala común. Cuando las variables presentan rangos diferentes, su importancia relativa varía y eso puede afectar

al desarrollo del modelo. La normalización corrige este desequilibrio homogeneizando todos esos datos, asegurando que todas acaben contribuyendo equitativamente al proceso de inferencia. De esta manera, se garantiza que, por ejemplo, las características representadas en un vector estén en la misma escala, lo que puede tener importancia para evitar que aquellos rasgos con valores numéricos más grandes dominen el modelo, pudiendo comprometer la precisión de la predicción.

Un ejemplo de normalización es la basada en la norma euclidiana (también conocida como normalización L_2 (González, 2023)). Esta se aplica a unos datos en forma de vector calculando su norma L_2 y dividiendo el vector por la misma norma para transformarlos en unitarios.

3.5.2. Codificadores

El codificador (*encoder*, en inglés) (Murel y Noble, 2024) es una red neuronal utilizada para el procesamiento de datos. Tal como sugiere su nombre, el codificador transforma una entrada dada en una representación vectorial compacta, capturando sus características esenciales en un espacio vectorial. Este actúa como función de proyección del *input* al espacio vectorial de representación. Se trata de un ejemplo de preprocesamiento de datos, ya que permite transformar esa información original en un vector más manejable que podría ser utilizado posteriormente en los modelos para un aprendizaje más enfocado en las características clave codificadas en ese espacio.

Los *encoders* más comunes se encuentran en el ámbito del Procesamiento de Lenguaje Natural (NLP, por sus siglas en inglés) donde suelen trabajar a partir de palabras. Sin embargo, es posible diseñar un codificador para cualquier tipo de dato siempre que se busque crear un espacio vectorial que capture entre sus distintas dimensiones las características relevantes del objeto de entrada. A continuación, se introduce el caso de los codificadores de imágenes (*Image Encoder*) que, como bien indica su nombre, trabajan con imágenes.

Las imágenes al final son matrices de píxeles, así que un codificador de imágenes podría extraer en esa representación vectorial características visuales relevantes presentes en la imagen. Estas podrían ser elementos generales (por ejemplo, formas o colores), así como objetos presentes (como personas, coches, árboles... y las relaciones espaciales entre ellos) entre otros. Como resultado, el *Image Encoder* generaría un vector en un espacio de dimensión d, que encapsularía toda esta información.

Un tipo de arquitectura usada en los *Image Encoders* son los recientemente introducidos *Vision Transformer* (ViT) (Dosovitskiy et al., 2020) que adaptan la arquitectura de los *Transformers* (originalmente usada para texto) al dominio visual.

Los transformadores (*Transformers*), en general, están diseñados para modelar relaciones entre *tokens* de entrada mediante un mecanismo conocido como *atención*. En tareas de procesamiento de lenguaje natural, los *tokens* suelen ser palabras o subpalabras. En el caso de los *vision transformers*, que operan sobre imágenes, el análisis directo a nivel de píxeles es inviable debido a su alto coste computacional, que crece cuadráticamente con el número de píxeles. Para abordar este problema, ViT divide la imagen de entrada en pequeños fragmentos de tamaño fijo llamados *patches*, los cuales se aplanan y proyectan linealmente a un espacio de dimensión

mayor. Cada uno de estos vectores proyectados es tratado como un *token*, de manera análoga a como los modelos de NLP manejan las palabras en una secuencia.

A cada *token* se le añade información de posición para mantener el contexto espacial original de la imagen. La secuencia resultante se introduce en un *Transformer* estándar, compuesto por múltiples capas de *self-attention*, donde cada *token* puede interactuar con todos los demás, permitiendo así capturar dependencias a largo alcance, independientemente de su proximidad espacial.

La salida final del ViT es una representación global de la imagen y sus características interrelacionadas, que se obtiene a partir de un *token* especial, el cual se concatena a la secuencia de parches y aprende a sintetizar la información relevante de toda la imagen y sus parches mediante el mecanismo de autoatención del *Transformer*.

Capítulo 4

Descripción del Trabajo

Este capítulo se centra en detallar el trabajo realizado a lo largo del proyecto, describiendo de la manera más completa posible todo lo desarrollado e investigado con el fin de ofrecer una comprensión global del proceso seguido. El contenido se estructura en varias secciones que se corresponden con las fases planteadas en el plan de trabajo (Sección 1.3).

Primero, se razonará la selección del conjunto de datos y se comentarán las ventajas y limitaciones que este posee. A su vez, se explorará una manera eficiente de almacenar y gestionar un volumen de datos tan grande.

Después, se explicará cómo se ha llevado a cabo la división del conjunto de datos en tres particiones complementarias (entrenamiento, validación y test) para poder usarlas respectivamente para entrenar los modelos, determinar sus configuraciones óptimas y evaluar su desempeño final.

A continuación, se definirán las arquitecturas, parámetros, funciones de pérdida y optimizadores disponibles para configurar los diferentes modelos. De esta manera, se podrán experimentar diferentes alternativas con el objetivo de encontrar la configuración que alcance el mejor rendimiento.

Seguidamente se enseñará cómo se ha realizado el entrenamiento de los modelos definidos monitoreando en todo momento el error producido por el conjunto de entrenamiento y validación. Asimismo, se supervisarán dichos errores para confirmar que se han evitado tanto el sobreajuste como el subajuste.

Posteriormente, se estudiará la información generada durante la evaluación de los diferentes modelos con el conjunto de test. Este estudio servirá para discutir el mejor prototipo basándose en el mejor desempeño en términos de precisión y capacidad de generalización ante imágenes no vistas.

Finalmente, se presentará la aplicación para usar el mejor modelo razonado anteriormente para predecir la ubicación de imágenes que cargues desde local. Además, se mostrará cómo funciona la aplicación y con un ejemplo se exhibirá cómo se ve su interfaz.

4.1. Selección y Preparación del Conjunto de Datos

En comparación con otras tareas de clasificación de imágenes, que generalmente se enfocan en un solo objeto, la geolocalización requiere detectar y combinar diversas características geográficas para identificar la localización. Algunas de estas características pueden ser: elementos naturales como la flora, el clima o los tipos de suelo; aspectos arquitectónicos y urbanos, como los estilos de construcción, las infraestructuras y las señales de tráfico (postes de luz, carreteras, bolardos); así como patrones de movilidad, como la conducción por la izquierda o la derecha en distintas regiones del mundo. Además, factores culturales, como el idioma o las costumbres, también pueden ser indicadores clave de ubicación. Todo esto se ve con mayor claridad en la Figura 4.1 donde vemos recuadradas en verde esas claves naturales, en rojo marcadores de tráfico como pueden ser señales o sentido de la conducción, en rosa un factor cultural muy determinante de la ubicación como es el idioma y en azul diferentes tipos de estilos arquitectónicos (todas estas imágenes son del *dataset* `OpenStreetView-5M`).

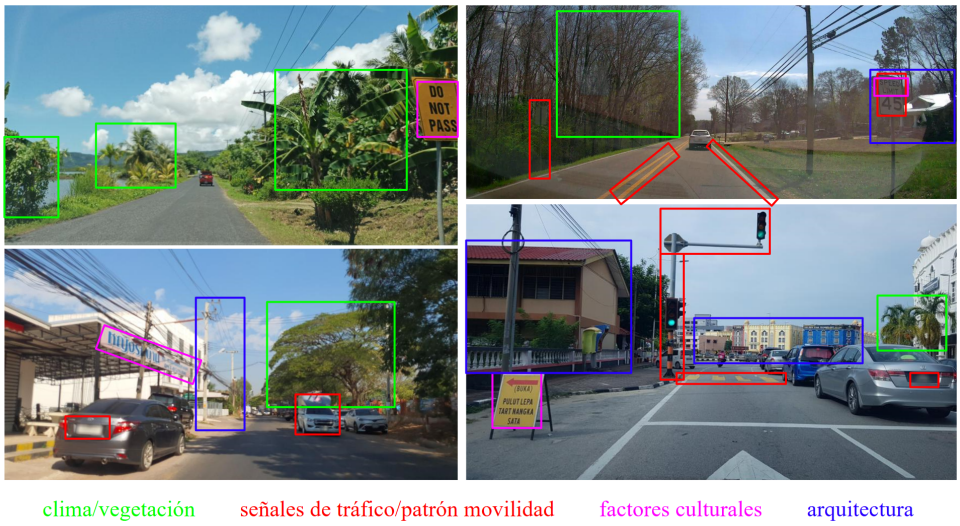

Figura 4.1: Características Visuales para la Geolocalización de una Imagen

Dada la complejidad de la tarea de geolocalización, es esencial elegir un conjunto de datos adecuado. El *dataset* debe tener un extenso conjunto de imágenes etiquetadas, ya que la red neuronal requiere de datos bien clasificados para generalizar correctamente. Predecir una coordenada GPS necesita de una representación rica tanto de la cultura como de la geografía de la Tierra. Por eso, el conjunto de datos, a su vez, debe estar bien distribuido a lo largo de toda la superficie terrestre para evitar sesgos geográficos que afecten a la precisión del modelo. Un conjunto de imágenes mal equilibrado puede resultar en un aprendizaje sesgado, en el cual la predicción de ciertas ubicaciones obtiene más peso, lo que compromete el objetivo de geolocalización generalizada.

A pesar de la existencia de abundantes imágenes geolocalizadas en Internet que podrían ser útiles para entrenar y evaluar modelos de visión, muy pocos enfoques supervisados han sido entrenados y evaluados específicamente para tareas de geo-

localización. Como bien introducen en el trabajo (Astruc et al., 2024), esto parece deberse a principalmente tres problemas: (i) muchos conjuntos de datos de acceso libre contienen una porción significativa de imágenes no localizables o de mala resolución que pueden introducir ruido en el modelo; (ii) los *datasets* de imágenes *street view*, que serían los más adecuados para la tarea, suelen ser de acceso restringido, propietarios o costosos de adquirir, mientras que las alternativas más asequibles suelen limitarse a ciertas ciudades o regiones geográficas; (iii) los conjuntos de datos tienden a estar muy sesgados, con imágenes principalmente de monumentos o de la cultura occidental, lo que puede inducir sesgos en la red neuronal.

Como mencioné en la Sección 1.1, para abordar los desafíos y limitaciones anteriormente mencionadas, decidimos usar el conjunto de datos `OpenStreetView-5M`. Este *dataset*, con su amplia cobertura geográfica y su estructura estandarizada, se presenta como una solución robusta e ideal para entrenar y evaluar los modelos, mitigando en gran medida los riesgos de sesgo y favoreciendo la generalización de entornos geográficos diversos.

4.1.1. Accesibilidad y Costo

`OpenStreetView-5M` es un extenso *dataset* de acceso abierto y gratuito que contiene 5,1 millones de imágenes *street view* de alta calidad, capturadas en diversas partes del mundo. Estas imágenes provienen de fuentes colaborativas como `Mapillary` (Mapillary, 2025) y están licenciadas bajo CC-BY-SA, lo que permite su uso libre siempre que se atribuya correctamente a los autores originales. El *dataset* ya está preparado para una validación simple, con una división predefinida en conjuntos de entrenamiento y test, y está disponible a través de `Huggingface` (Astruc et al., 2024).

4.1.2. Localizabilidad

Como se detalla en (Izbicki et al., 2020) y en (Astruc et al., 2024), las imágenes presentan un rango de localizabilidad que se considera una propiedad inherente y perceptual. Véanse los ejemplos en la Figura 4.2. Las imágenes no localizables (*non-localizable*, en inglés) son aquellas que carecen de información suficiente para asociarlas a una ubicación específica o tienen una resolución tan baja que no permiten un análisis adecuado. Las imágenes débilmente localizables (*weakly localizable*, en inglés) contienen solo pistas indirectas o imprecisas sobre su localización, como pueden ser personas, animales u objetos en escenas interiores. Por otro lado, las imágenes localizables (*localizable*, en inglés) son aquellas que incluyen la información necesaria para permitir una estimación relativamente precisa de su ubicación, como ocurre con la mayoría de imágenes *street view*, que poseen características geográficas identificables como el clima, la naturaleza y la arquitectura entre otras. En el extremo del espectro se encuentran las imágenes fuertemente localizables (*strongly localizable*, en inglés), que contienen toda la información necesaria para ser geolocalizadas con exactitud. Ejemplos de estas imágenes incluyen monumentos emblemáticos y paisajes icónicos, cuya localización puede ser instantáneamente identificada por la mayoría de las personas.

Según los creadores de `OpenStreetView-5M`, se estima que el 96,1 % (± 0,57 %) de las imágenes del conjunto de datos son localizables con un intervalo de confianza del 95 %. Dentro de las imágenes no localizables y las débilmente localizables, el 70 % (2,7 % del total) presentan baja calidad, como subexposición o sobreexposición (debido a una falta o exceso de luz al momento de tomar la fotografía), desenfoque o rotación. El 30 % restante (1,2 % del total) se caracteriza por un mal encuadre o porque son imágenes sacadas en interiores o en túneles.

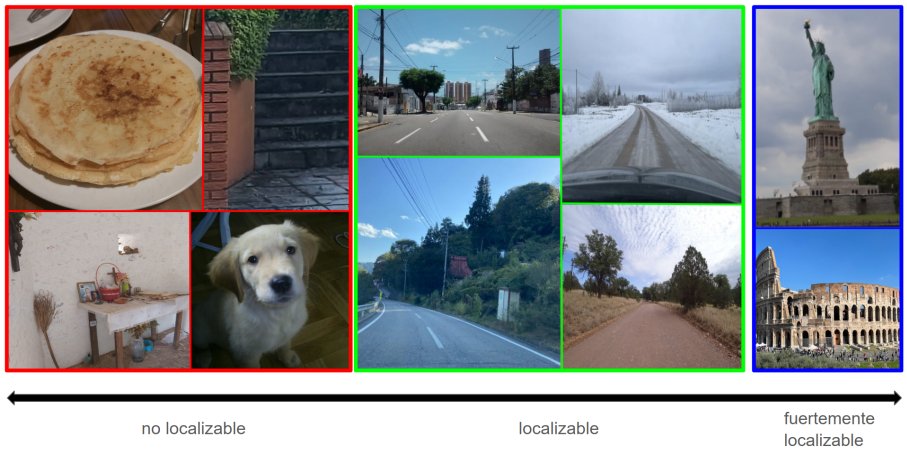

Figura 4.2: Espectro de Localizabilidad

4.1.3. Distribución Geográfica Equilibrada y Mitigación de Sesgos

El *dataset* `OpenStreetView-5M` ha sido diseñado para maximizar la diversidad y riqueza de los datos y para minimizar los sesgos geográficos, siendo ambos problemas que mencionaba que habitualmente pasaban en otros conjuntos de datos de geolocalización. A continuación, se detallan los principales aspectos que garantizan esa distribución equilibrada y las estrategias utilizadas para evitar otros desafíos no considerados.

4.1.3.1. Escala y Alcance

`OpenStreetView-5M` está compuesto por 4 894 684 imágenes de entrenamiento y 210 122 imágenes de test, asegurándose así una base de datos suficientemente extensa para el desarrollo de los modelos robustos que estoy planteando.

Muchos *datasets* están restringidos a unas pocas ciudades o presentan un fuerte sesgo hacia la cultura occidental. Sin embargo, las imágenes de `OpenStreetView-5M` están uniformemente distribuidas a lo largo de la superficie terrestre abarcando 70 000 ciudades y 225 países y territorios. Todo esto se puede ver en la Figura 4.3. La diversidad geográfica del conjunto de test tiene una entropía normalizada de 0,78 (Bürkner et al., 2024), lo que indica una alta heterogeneidad y uniformidad en la distribución de las imágenes. El conjunto de entrenamiento, por su parte, tiene una entropía normalizada del 0,67, valor comparable con la entropía de la distribución

del área de los países (0,71), lo que sugiere un equilibrio suficientemente adecuado en la representatividad de las regiones.

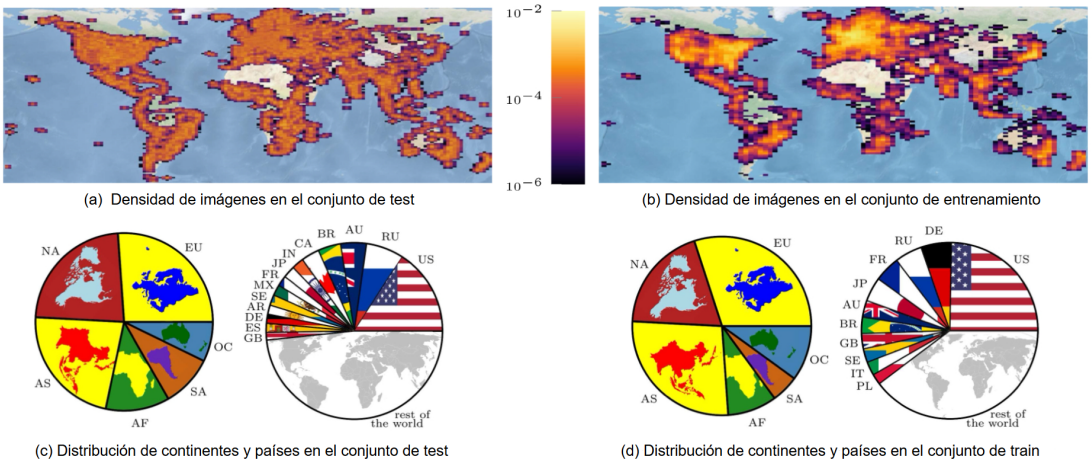

Figura 4.3: Distribución geográfica de `OpenStreetView-5M` (Astruc et al., 2024)

4.1.3.2. Separación Espacial y Separación de Secuencias

Sin imponerse una separación cuidadosa entre las imágenes de entrenamiento y test, la tarea de geolocalización podría verse reducida a un problema de reconocimiento de lugares en lugar de evaluar la capacidad del modelo de aprender a identificar patrones geográficos generales. Para evitar este problema, `OpenStreetView-5M` se asegura que ninguna imagen del conjunto de entrenamiento se encuentre a menos de 1 km de una imagen del conjunto de test.

Las imágenes *street view* suelen capturarse mediante una cantidad limitada de cámaras montadas en la parte delantera o trasera de vehículos que recorren una determinada región. Esto genera una correlación entre la ubicación, los vehículos y las cámaras, lo que podría explotarse para simplificar la tarea de geolocalización. Un ejemplo de este fenómeno lo tenemos en el juego de `GeoGuessr` (GeoGuessr, 2025) donde existe un *geotip* usado ampliamente por los jugadores profesionales para identificar el país de Ghana. En este caso, el país puede reconocerse porque el vehículo `Google Street View` empleado para cartografiar el país tiene una cinta adhesiva negra en la barra delantera del portaequipajes (GeoTips, 2025). Para mitigar este sesgo, `OpenStreetView-5M` impone la restricción de que ninguna secuencia de imágenes (es decir, una serie continua de imágenes capturadas por el mismo usuario en `Mapillary`) esté presente en ambos conjuntos de entrenamiento y test. Si bien esto no impide por completo que imágenes capturadas por el mismo vehículo en días distintos se repitan en ambos conjuntos, sí reduce esta posibilidad.

4.1.3.3. Construcción del Dataset

Para evitar que las regiones con mayor densidad de imágenes estén sobrerrepresentadas, se definió una cuadrícula de 100x100 metros a nivel global y se seleccionó

aleatoriamente una imagen por celda. Posteriormente, tanto el conjunto de entrenamiento como el de test se muestrearon con una probabilidad proporcional a la densidad local de imágenes elevada a la potencia de $-0{,}75$. Este enfoque busca equilibrar el muestreo basado en densidad (que tiende a favorecer los centros urbanos) y el muestreo basado en áreas (que podría favorecer países con mayor superficie). Finalmente, para garantizar una separación espacial y temporal efectiva, se eliminaron del conjunto de test las imágenes que se encontraban dentro de un radio de 1 km de cualquier imagen de entrenamiento o que compartiesen un ID de secuencia.

4.1.3.4. Limitaciones

A pesar de que el *dataset* `OpenStreetView-5M` consigue mitigar en gran medida los problemas que he comentado, sigue presentando ciertas limitaciones:

(i) Sesgo Geográfico en el conjunto de entrenamiento. No se ha podido eliminar por completo la dependencia del conjunto de entrenamiento respecto a las contribuciones de los usuarios de `Mapillary`, lo que provoca un sesgo persistente hacia regiones occidentales y norteamericanas. No obstante, el conjunto de test ha sido diseñado específicamente para corregir este desequilibrio, asegurando una distribución más equitativa y permitiendo una evaluación del modelo prácticamente libre de sesgos geográficos.

(i) Separación de usuarios. Ya se ha comentado que se logró garantizar que las imágenes de una misma secuencia no estuvieran en ambos conjuntos de entrenamiento y test. Sin embargo, debido a la falta de metadatos en el momento de la construcción del *dataset*, no se pudo evitar la inclusión de imágenes capturadas por el mismo usuario en diferentes días.

(i) Resolución. Las imágenes del *dataset* tienen una altura de 512 píxeles y una anchura media de 792 ± 127 píxeles. Esto restringe parcialmente la posibilidad de hacer zoom y leer textos distantes, como los que se encuentran en señales de tráfico o nombres de calles, potencialmente reduciendo algunas pistas visuales clave para la geolocalización.

4.1.4. Almacenamiento y Acceso del Conjunto de Datos

El *dataset* utilizado para entrenar y evaluar el modelo contiene una cantidad masiva de imágenes, poco más de 5,1 millones. Debido a esto, la preparación del conjunto de datos para encontrar una manera eficiente de almacenarlo, acceder a él y gestionarlo es esencial, ya que se utilizará en las fases posteriores del proyecto, y es necesario evitar que se convierta en un cuello de botella.

Inicialmente, me enfrenté a dos problemas principales: (i) ¿dónde almacenar todas las imágenes, que en su versión comprimida suman aproximadamente 246 GB para el conjunto de entrenamiento y 9,5 GB para el conjunto de test? y (ii) ¿cómo acceder a ellas de manera rápida y eficiente para visualizarlas, buscar e iterar entre ellas y procesarlas?

Dado que los datos se encuentran en `Huggingface`, la primera aproximación fue acceder al conjunto de entrenamiento y test utilizando la librería `datasets` de *Hugging Face* (Hugging Face, 2025). Esta librería permite aprovechar la capacidad

de carga en *streaming*, es decir, los datos no se descargan todos de golpe desde un *dataset* remoto, sino que se procesan en tiempo real bajo demanda, evitando la necesidad de almacenarlos en disco. Esta estrategia solucionó el primer problema, ya que la capacidad de almacenamiento de mi ordenador es limitada (237 GB en un SSD rápido, pero con poco espacio, y 931 GB en un HDD más lento pero con mayor capacidad). Además, cargarlos de esta forma permitió acceder a las imágenes junto con sus metadatos, lo que facilitaba el acceso. Sin embargo, iterar sobre las imágenes usando esta librería en modo *streaming* resultó ser ineficiente y lento. Esto podría deberse a que cargar los datos bajo demanda requiere una conexión a internet rápida y estable, y las solicitudes HTTP añaden latencia, lo que provoca un cuello de botella y hace que el proceso sea tan pesado que fue necesario buscar una solución más rápida.

Decidí entonces sacrificar parte del problema de espacio y almacenar todas las imágenes localmente, ya que el acceso directo, a priori, podría ser mucho más rápido sin necesidad de cargar los datos bajo demanda. Además, podía aprovechar la mayor velocidad del SSD para procesar los conjuntos de imágenes en lotes porque su capacidad limitada impedía almacenar el *dataset* completo. De esta forma, incrementé considerablemente el uso de espacio a cambio de ganar rapidez y eficiencia. No obstante, pronto descarté la opción de usar la SSD como almacenamiento parcial del *dataset* ya que en fases posteriores necesitaría trabajar con el conjunto completo, y gestionar el almacenamiento eliminando archivos a medida que los voy necesitando sería poco práctico. Por ello, comencé a descargar todos los archivos *zip* en HDD. Esta solución, sin embargo, no resolvía completamente el problema de eficiencia, ya que el HDD es considerablemente más lento que el SSD. Además, me encontré con una nueva dificultad: una vez descargado el archivo *zip*, la descompresión de este tardaba bastante tiempo. Sumándole esto al tiempo que ya se había necesitado para descargar el *zip*, esto aumentaba notablemente la espera necesaria para disponer de todas las imágenes. Debido a esto, descarté la idea de tener todas las imágenes descomprimidas y opté por seguir utilizando los archivos *zip*.

Un cuarto problema surgió al trabajar con las imágenes en bruto, ya que no están asociadas con las coordenadas ni otros metadatos del conjunto de datos de `OpenStreetView-5M`. Era necesario encontrar una forma de vincular las imágenes con sus metadatos, sabiendo que el identificador único de cada imagen se encontraba en el nombre del archivo *jpg*.

Mi solución llegó cuando encontré `LMDB` (Lightning Memory-Mapped Database) (Symas Corporation, 2025) y su implementación en Python (Watson, 2025). `LMDB` es una base de datos clave-valor extremadamente rápida y eficiente en el uso de memoria, desarrollada originalmente para el proyecto `OpenLDAP` (The OpenLDAP Project, 2025). Gracias a su uso de archivos mapeados en memoria, `LMDB` ofrece el rendimiento de lectura de una base de datos completamente en memoria, mientras conserva la persistencia de las bases de datos tradicionales basadas en disco. Además, se usa ampliamente en aplicaciones de alto rendimiento, como bases de datos embebidas, Inteligencia Artificial y almacenamiento de grandes volúmenes de datos, lo que la convierte en una solución perfecta para el problema. De esta forma, pude crear una base de datos `LMDB` usando los archivos *zip*, siendo las claves los IDs de las imágenes y los valores las propias imágenes. Si bien el proceso inicial tomaba algo

más de tiempo, el beneficio posterior en velocidad de acceso lo hizo valer la pena. A su vez, este acceso optimizado permitió almacenar todo en el disco HDD sin comprometer el rendimiento, resolviendo así el problema de espacio. Aunque el HDD es más lento que el SSD, la eficiencia de `LMDB` en la gestión de memoria compensaba esta diferencia, garantizando un acceso rápido a las imágenes y evitando el desafío abierto que habíamos dejado antes usando HDD. Al convertir el conjunto a `LMDB`, su tamaño final fue de 245 GB para entrenamiento y 12,5 GB para test, algo imposible de manejar originalmente en el SSD debido a sus limitaciones de capacidad. Asimismo, la base de datos, al ser clave-valor, permitió asociar cada imagen con su identificador único, acercándonos aún más a una organización conjunta de los datos y metadatos. Un resumen de todas las alternativas comentadas se puede ver en la Tabla 4.1

Idea	Almacenamiento	Eficiencia
Hugging Face	✓ No ocupa espacio	✗ No eficiente
SSD	✗ Ocupa espacio (no hay suficiente)	✓ Rápido
HDD	✗ Ocupa 255,5 GB	✗ Lento
LMDB	✗ Ocupa 257,5 GB	✓ Rápido

Tabla 4.1: Preparación conjunto de datos

Al final, la solución `LMDB` resolvió todos los problemas con gran eficacia. Aunque fue necesario almacenar todo localmente, ocupando cerca de 260 GB, logré mantenerlo en HDD sin comprometer la eficiencia de acceso, lo que además permitió solucionar el segundo problema.

Por otro lado, logré generar toda la base de datos `LMDB` sin necesidad de descomprimir ningún archivo, evitando así tiempos de procesamiento adicionales. Finalmente, la organización clave-valor facilitaba la asociación de las imágenes con sus metadatos a través de sus IDs ya que, mediante una ordenación eficiente de IDs tanto en la base de datos como en los metadatos, puedo utilizarlos de manera conjunta sin complicaciones.

4.2. División del Conjunto de Datos

Como se detalló en el plan de trabajo (ver Sección 1.3), el conjunto de datos se dividirá en tres subconjuntos complementarios: entrenamiento, validación y test. Esta partición, introducida en la Sección 3.2, es una práctica habitual en el desarrollo de modelos, ya que permite evaluar distintas configuraciones y seleccionar la más adecuada. El conjunto de entrenamiento se usará para entrenar el modelo, el de validación servirá para determinar las configuraciones óptimas y el de test se empleará para medir el rendimiento final de los modelos.

4.2.1. División Entrenamiento-Test

Inicialmente, como se mencionó en la Sección 4.1.3, `OpenStreetView-5M` está previamente dividido en un conjunto de entrenamiento y un conjunto de test, teniendo una división de 210 122 imágenes (4,116 % del total) destinadas a evaluación y 4 894 684 imágenes (95,884 % del total) para entrenamiento.

Esta partición se creó garantizando un conjunto de test diverso, heterogéneo y uniformemente distribuido geográficamente, permitiendo una evaluación del modelo prácticamente libre de sesgos. Aunque a primera vista la porción del test podría parecer insuficiente, el *dataset* en su totalidad cuenta con muchos datos: 5 104 806 imágenes, lo que significa que, incluso con un $\approx 4{,}1\,\%$, se dispone de más de 200 000 ejemplos y es suficiente para obtener métricas estables. Además, dado que el conjunto de test fue diseñado cuidadosamente para reflejar la distribución real geográfica de los datos, su representatividad no se ve comprometida a pesar del porcentaje relativamente pequeño, por lo que no es necesario ampliarlo.

4.2.2. Conjunto de Validación

En aras de evitar que los modelos caigan en un sobreajuste o subajuste (ver conceptos en la Sección 3.3) de los datos de entrenamiento, se va a aplicar la técnica de *early stopping*, descrita en la Sección 3.4.1, para determinar cuándo detener el proceso de entrenamiento.

El conjunto de test no debe utilizarse en el *early stopping* bajo ninguna circunstancia, ya que su propósito es servir como una evaluación completamente independiente del modelo. Si se introduce en la decisión de parada, se estaría utilizando indirectamente un conjunto que debería permanecer intacto para la evaluación final, comprometiendo así la objetividad de los resultados. En otras palabras, el conjunto de test dejaría de ser un reflejo realista del rendimiento del modelo en datos no vistos, ya que habría influido en la optimización de un hiperparámetro, como, por ejemplo, el número de épocas de entrenamiento.

Por esta razón, es fundamental disponer de un conjunto de validación para aplicar el *early stopping* y que actúe como referencia durante el entrenamiento, permitiendo ajustar estos hiperparámetros sin afectar la evaluación final del modelo. Consiguiendo así reservar el buen equilibrio, heterogeneidad y equidad del conjunto de test para una evaluación imparcial del modelo.

4.2.2.1. Construcción del Conjunto de Validación

Para construir el conjunto de validación, es importante definir qué datos utilizar. Por un lado, está el conjunto de test y, por el otro, el conjunto de entrenamiento. La opción de fusionarlos y volver a dividirlos en tres conjuntos disjuntos queda completamente descartada, puesto que se incurriría en los problemas señalados en la Sección 4.1.3.4 sobre las limitaciones del conjunto de entrenamiento. Asimismo, esto comprometería el esfuerzo dedicado en `OpenStreetView-5M` en la construcción de un conjunto de test equilibrado y libre de sesgos, como se explicó en la Sección 4.1.3.3.

Tampoco es adecuado generar el conjunto de validación a partir del conjunto

de test, ya que reduciría la cantidad de datos disponibles para la evaluación final y afectaría a la diversidad con la que fue diseñado.

Dado que `OpenStreetView-5M` proporciona un conjunto de entrenamiento suficientemente amplio, una buena estrategia sería dividirlo en dos subconjuntos: uno para validación y otro con una proporción mayor, para el entrenamiento, que es la opción más habitual en estos casos. Es crucial que el conjunto de entrenamiento mantenga la mayor cantidad de datos posible para que la red neuronal aprenda de manera más efectiva.

Se empleará un enfoque sencillo, pero que garantice la preservación de la distribución geográfica dentro del conjunto de entrenamiento. Si se hace una partición sin cuidado, escogiendo de manera aleatoria imágenes del conjunto de entrenamiento sin considerar su ubicación, podría generarse un desbalance geográfico, lo que llevaría a una representación insuficiente de ciertas regiones del mundo para la fase de entrenamiento de la red neuronal. Para mitigar este problema, se divide la superficie terrestre en cuadrículas de $\frac{1}{4}$ de grado de latitud por $\frac{1}{4}$ de grado de longitud, como se muestra en la Figura 4.4. Una vez establecida esta segmentación, la partición en conjuntos de entrenamiento y validación se llevará a cabo de manera estratificada, asegurando que la distribución geográfica en ambos conjuntos sea lo más equilibrada posible.

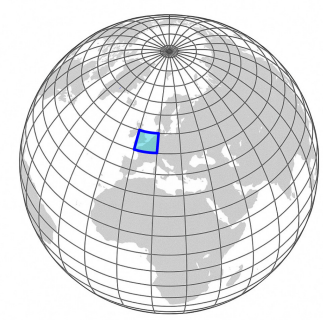

Figura 4.4: División cuadriculada de la Tierra

Veamos en kilómetros cuánto abarca cada una de estas cuadrículas. Como primera aproximación, consideraremos la Tierra como una esfera, aunque en realidad es un esferoide oblato.

Bajo esta suposición, el intervalo de $\frac{1}{4}$ de grado de latitud se mantiene prácticamente constante en todas las longitudes y equivale a aproximadamente 27,83 km. Esto se debe a la propiedad de los meridianos, que pueden entenderse como "semi-ecuadores" verticales que se extienden desde un polo hasta el otro y su valor de longitud β se corresponde con el ángulo horizontal con respecto al Meridiano de Greenwich. De esta manera, las longitudes toman valores entre $-180°$ hacia el oeste de ese meridiano y $180°$ hacia el este. Mientras tanto, las latitudes corresponden a circunferencias horizontales a diferentes alturas, determinadas por el ángulo α que forma el plano horizontal que contiene al ecuador con respecto al vector con origen en el centro de la Tierra y destino en un punto de la superficie a latitud α. Debido a esta configuración, las latitudes varían desde $-90°$ en el polo sur, pasando por $0°$ en el ecuador, hasta $90°$ en el polo norte. En la Figura 4.5 se pueden observar los me-

ridianos como semicircunferencias, la disposición de los paralelos y como funcionan los ángulos que representan las latitudas y las longitudes.

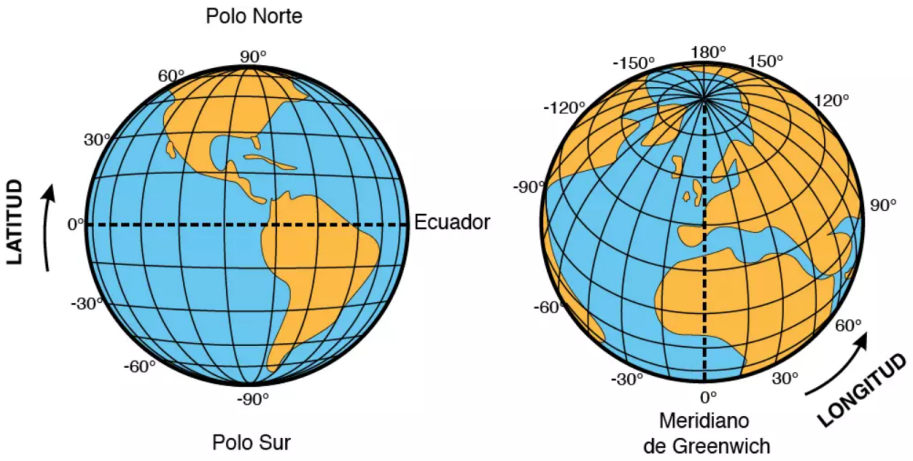

Figura 4.5: Latitud-Longitud vs Meridianos-Paralelos (Proyecto Viajero, 2024)

Gracias a esta estructura, la superficie esférica no introduce distorsión en la distancia a lo largo de los meridianos y, por lo tanto, tampoco en la medición de la latitud. El valor de 27,83 km proviene del hecho de que la longitud de la línea del ecuador es de 40 075 km. En una esfera, todas las circunferencias trazadas sobre su superficie cuyo centro coincida con el centro de la esfera tienen la misma longitud. Considerando la aproximación ya mencionada de tratar a la Tierra como una esfera, dado que los meridianos son semicircunferencias verticales que conectan los polos y cuyo centro es el centro terrestre, su longitud equivale a la mitad de la del ecuador. Finalmente, como la latitud abarca un total de 180° $[-90°, 90°]$, al dividirla en cuartos de grado, se obtiene:

$$\frac{\text{mitadLongEcuador}}{\text{cuartoGradoLat}} = \frac{\frac{40075}{2}}{180 \cdot 4} \approx 27{,}83 \text{ km}$$

Este resultado garantiza que cada cuadrícula definida mantenga una representación espacial uniforme en términos de latitud.

Sin embargo, el intervalo de $\frac{1}{4}$ de grado en la longitud va disminuyendo a medida que aumenta la latitud, ya que, por ejemplo, en los polos todos los meridianos convergen. La fórmula que describe esta variación es:

$$\frac{2 \cdot \pi \cdot R \cdot \cos(\theta)}{360 \cdot 4} \tag{4.1}$$

Donde R es el radio de la Tierra y θ es el grado de latitud. En el ecuador, por ejemplo, la longitud correspondiente a $\frac{1}{4}$ de grado es aproximadamente 27,79 km, mientras que a una latitud de 45°, esa distancia se reduce a 14,6 km (como puede observarse, se va reduciendo a medida que nos acercamos a los polos).

Ahora, vamos a demostrar la fórmula mencionada apoyándonos en la Figura 4.6. Consideremos el triángulo rectángulo formado por los puntos A, B y C. Sabemos que los puntos correspondientes a una latitud θ se encuentran en una circunferencia de centro P y radio r, y utilizando las razones trigonométricas de los triángulos

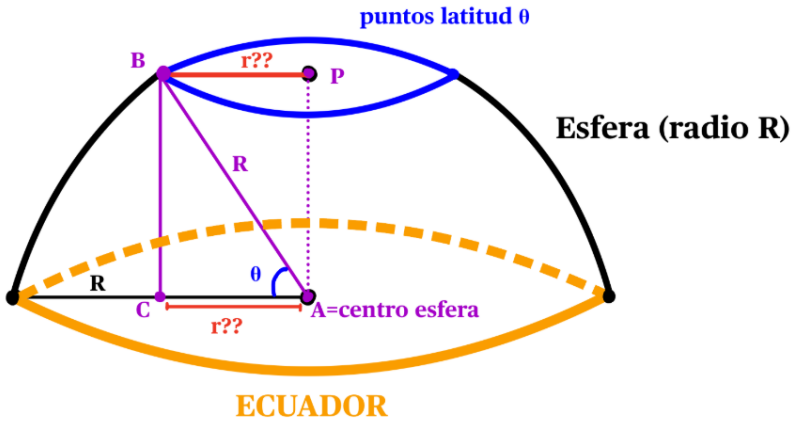

Figura 4.6: Longitud variable

rectángulos, podemos calcular: r = hipotenusa $\cdot \cos(\theta)$. Donde la hipotenusa, al ser el lado del triángulo que va desde el centro de la esfera hasta su superficie, se corresponde con el radio de la esfera R (en nuestro caso, el radio promedio de la Tierra es de 6370 km). Una vez que hemos calculado ese radio r, para determinar cuántos kilómetros abarca el arco correspondiente a $\frac{1}{4}$ de grado de longitud en una determinada latitud, simplemente calculamos el perímetro del paralelo a esa latitud y la dividimos entre 360·4 (ya que la longitud total se divide entre 360 grados y, como estamos trabajando con cuartos de grado, debemos considerar esa subdivisión). La fórmula resultante justo la que hemos representando en (4.1):

$$\frac{\text{longParaleloLatX}}{\text{cuartoGradoLong}} = \frac{2{\cdot}\pi{\cdot}\text{radioCircunfLatX}}{\text{cuartoGradoLong}} = \frac{2{\cdot}\pi{\cdot}R{\cdot}\cos(\theta)}{360{\cdot}4}\ \text{km}$$

Una vez dividida la superficie terrestre en una cuadrícula, realizamos la partición del conjunto de datos de manera estratificada. Para ello, juntamos cuantas imágenes de entrenamiento hay en cada celda y las distribuimos siguiendo una proporción del 82 % para entrenamiento y 18 % para validación. Como resultado, se generan 37 824 clases distintas representando celdas que contienen al menos una imagen en la superficie terrestre.

Analizando los valores extremos de latitud y longitud en el conjunto de entrenamiento (véase Tabla 4.2), observamos que no hay imágenes en latitudes inferiores a $-55°$ ni superiores a 79°. Esto implica una ausencia de 35 grados en el hemisferio sur (90° - 55°) y 11 grados en el hemisferio norte (90° - 79°). En cuanto a la longitud, no hay representación por debajo de $-177°$ ni en 179°, lo que deja 3 grados sin cobertura. Si consideramos una cobertura ideal de la superficie terrestre, excluyendo estas zonas no representadas, el número total de celdas sería de $(180 - 46) \cdot 4 \cdot (360 - 4) \cdot 4 = 763264$. Sin embargo, dado que aproximadamente el 70 % de la superficie terrestre es agua, solo el 30 % contendría imágenes, lo que reduciría la estimación a $763\,264 \cdot 0{,}3 \approx 228\,979$. Véase Figura 4.7, donde se ve claramente esa cobertura teórica e ideal de las imágenes del conjunto de entrenamiento ya que se ha tachado en rojo lo descartado por límite del conjunto de entrenamiento y en azul lo descartado por el agua. A pesar de esto, el número real de celdas con imágenes es significativamente menor. Esto se debe a la falta de representación en

extensas regiones del conjunto de entrenamiento, como el centro de África, el desierto del Sahara, la tundra de Rusia y el oeste de China, como se muestra en la Figura 4.3.

	Min	Max
Latitud	−54,887208333333	78,655882045086
Longitud	−176,81130912065618	178,7223890581564

Tabla 4.2: Límites Conjunto de Entrenamiento

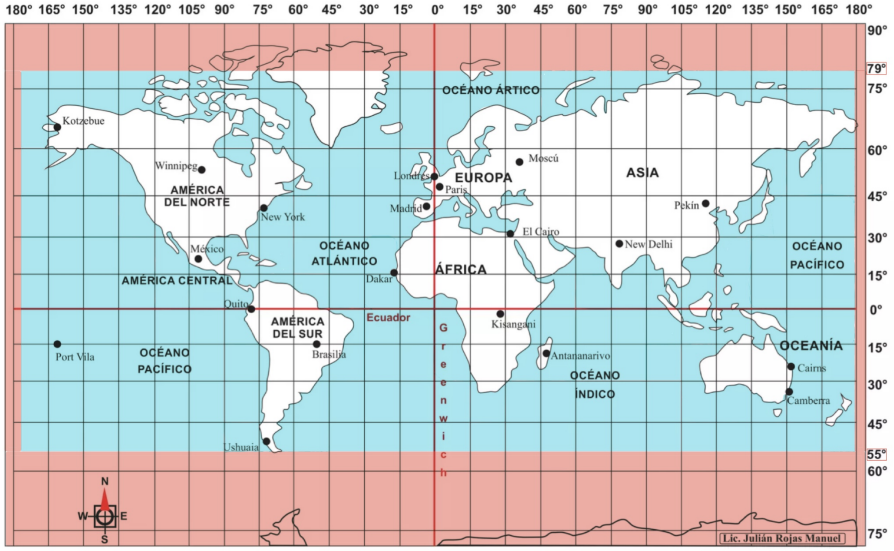

Figura 4.7: Cobertura teórica del conjunto de entrenamiento (Manuel, 2016)

Por otro lado, existen 2432 grupos que contienen únicamente un elemento, lo que impide su división manteniendo la proporción establecida. Dado que cada partición del conjunto de datos tiene un propósito específico (entrenamiento vs. validación para *early stopping* y evitar el sobreaprendizaje), resulta más adecuado asignar estas imágenes directamente al conjunto de validación. De este modo, el modelo no adquirirá conocimiento superfluo de esas coordenadas. Estos grupos se separan previamente y se incorporan al conjunto de validación. Para el resto de los datos de entrenamiento, aplicamos la división estratificada mencionada del 82 %-18 %, asegurando así un conjunto de validación adecuado para la aplicación del *early stopping*.

4.2.2.2. Conclusiones

En conclusión, disponer de un conjunto de validación es crucial para llevar a cabo un entrenamiento evitando sobreajuste sin comprometer la imparcialidad del conjunto de test. Al final se ha construido fraccionando el conjunto de entrenamiento en dos y plantear una partición estratificada ayuda a mantener un balance geográfico asegurando que todas las regiones del mundo estén representadas tanto en el conjunto de entrenamiento como en el de validación. De esta manera, nos ha terminado

quedando una separación de 4 011 646 imágenes para entrenamiento (78,586 % del total) y 883 038 imágenes para validación (17,298 % del total).

Una vez finalizado el entrenamiento tras la aplicación del *early stopping*, se analizarán las gráficas del error de entrenamiento y validación cometidos en cada época para confirmar si el modelo está subajustado o sobreajustado. El *early stopping* ayuda a prevenir que la tasa de error en el conjunto de entrenamiento se desvíe significativamente de la de validación y será eso justamente lo que se buscará en ese análisis de gráficas mencionado. Esto garantizará que el modelo esté generalizando adecuadamente y no se haya sobreajustado a los datos de entrenamiento.

4.3. Definición de los Modelos y sus Parámetros

En este apartado se irán definiendo cada uno de los componentes principales (arquitecturas, parámetros clave, funciones de pérdida y optimizadores) que configuran los modelos que se van a entrenar para este proyecto. Estos son fundamentales para experimentar con diferentes alternativas en busca de alcanzar un rendimiento óptimo y preciso en la tarea de geolocalización. Las elecciones adecuadas de estas estructuras, junto con los ajustes óptimos de estos parámetros relevantes, la selección de buenas funciones de pérdida y el uso de técnicas de optimización, son cruciales para el éxito de los modelos. Finalmente, se detallarán las respectivas características que tienen los diferentes prototipos que se van a desarrollar.

4.3.1. Arquitectura de los Modelos

El objetivo principal del modelo a desarrollar es determinar con la mayor precisión posible la ubicación geográfica de una imagen a partir de las pistas geográficas que ésta contiene. El modelo usará las representaciones internas, aprendidas durante el entrenamiento, que capturan características visuales distintivas relacionadas con distintas regiones del mundo.

Con el fin de abordar esta tarea, la arquitectura propuesta se estructura en dos módulos principales, cada uno con una función específica dentro del proceso de inferencia:

- Codificador de Imágenes (*Image Encoder*, en inglés) $f^{\text{img}} : \mathcal{I} \longrightarrow \mathbb{R}^d$: este componente transforma cada imagen de entrada $\mathcal{I}$ en una representación vectorial de dimensión d. Dicho vector busca capturar de manera compacta las características visuales relevantes de la imagen, facilitando su procesamiento en etapas posteriores.

- Módulo de Geolocalización $f^{\text{coord}} : \mathbb{R}^d \longrightarrow \mathcal{C}$: utilizando la representación vectorial generada por el codificador de imágenes, este segundo módulo, implementado como una red neuronal, lleva a cabo una regresión que predice las coordenadas geográficas asociadas a la imagen, es decir, su latitud y longitud.

Antes de pasar al modelo de geolocalización, el vector de características generado por el *Image Encoder* se normaliza. La normalización (ver Sección 3.5.1) es un

paso crucial en el preprocesamiento de datos que asegura que las características del vector estén en la misma escala y evita que aquellas con valores numéricos más grandes dominen el modelo. A priori se asume que todas las características que pueda contener el vector que representa la imagen contribuyen de manera similar a la predicción final de su ubicación y es por eso que es necesaria una normalización.

En este caso, se aplica la normalización basada en la norma euclidiana (ejemplo de norma presentado en Sección 3.5.1) sobre los vectores de características generados por el *Image Encoder*. De esta manera, se asegura que cada vector quede con una longitud unitaria representando únicamente su dirección en el espacio latente. Esto ayuda a la comparación y la búsqueda de similitudes entre las características, algo que el Módulo de Geolocalización tendrá que hacer en el entrenamiento y de esta forma le facilitamos la tarea.

Este paso no solo mejora el rendimiento y la estabilidad del modelo, sino que también facilita la interpretación y visualización de los datos, al asegurar que todas las características se encuentren en la misma escala.

4.3.2. Codificador de Imágenes

El Codificador de Imágenes (ver Sección 3.5.2) constituye el primer componente fundamental del modelo propuesto. Su propósito es preprocesar la imagen de entrada generando una representación vectorial compacta y significativa, que sirva como base para el posterior tratamiento por parte del Módulo de Geolocalización. La función principal de este módulo debe ser extraer características visuales relevantes presentes en la imagen, como por ejemplo elementos naturales (flora, clima o suelo), o aspectos arquitectónicos y urbanos (estilos de construcción, infraestructuras), codificándolas en un vector de dimensión d que encapsularía toda esta información.

En este trabajo se ha optado por utilizar `CLIP` (*Contrastive Language-Image Pre-training*) (Radford et al., 2021), un modelo de red neuronal desarrollado por `OpenAI`. Se trata de un modelo multimodal cuyo objetivo es establecer una conexión entre imágenes y texto mediante un proceso de preentrenamiento basado en aprendizaje contrastivo. Este enfoque busca capturar relaciones semánticas entre descripciones textuales e imágenes, maximizando la similitud entre aquellas que se corresponden y minimizándola en los pares que no guardan relación.

Durante el entrenamiento, dado un lote de N pares (imagen, texto), `CLIP` se entrena para identificar cuáles de las $N \times N$ combinaciones posibles son las correctas. Para ello, aprende un espacio de incrustación multimodal en el que se proyectan conjuntamente las representaciones de imagen y texto. Esto se logra mediante el entrenamiento simultáneo de un codificador de imágenes y un codificador de texto, optimizados para maximizar la similitud del coseno entre las incrustaciones de los N pares reales del lote, y minimizar dicha similitud para las $N^2 - N$ combinaciones incorrectas.

Como resultado, ambos codificadores proyectan sus respectivas entradas en un espacio latente común, donde imágenes y textos semánticamente relacionados se ubican en posiciones cercanas, mientras que aquellos que no guardan relación se alejan entre sí. La Figura 4.8 ilustra la arquitectura general de `CLIP`, compuesta por dos redes neuronales especializadas: una para el procesamiento de imágenes y otra

para el de texto.

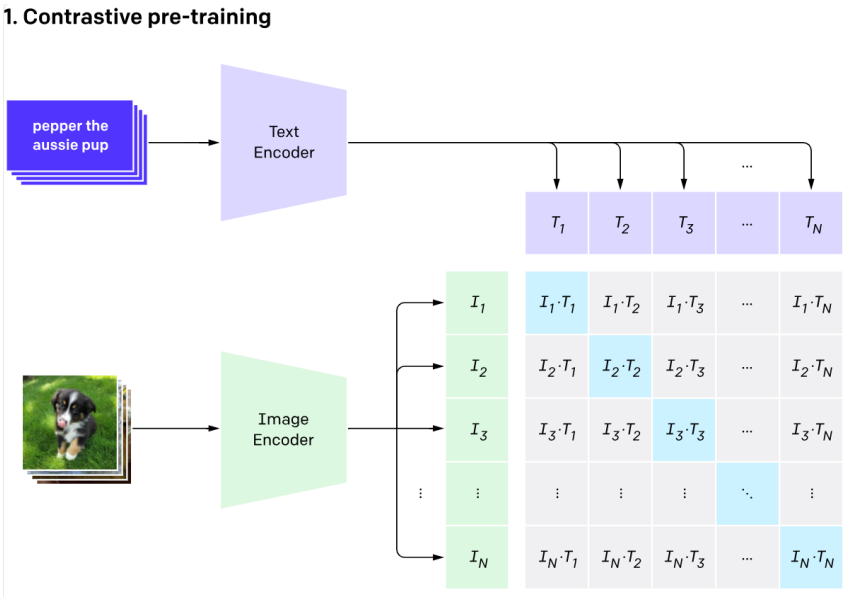

Figura 4.8: Estructura CLIP (Radford et al., 2021)

`CLIP` se entrena utilizando una gran variedad de imágenes acompañadas de descripciones en lenguaje natural, asociando cada imagen con su correspondiente texto descriptivo. Para ello, se construyó un conjunto de datos de entrenamiento compuesto por 400 millones de pares (imagen, texto), recolectados a partir de una gran variedad de fuentes públicas disponibles en Internet.

Gracias a la diversidad de este conjunto de entrenamiento, el enfoque de aprender usando lenguaje natural y al empleo de un esquema de aprendizaje contrastivo, `CLIP` es capaz de reconocer una amplia gama de conceptos visuales y textuales, permitiéndole realizar tareas sin haber sido entrenado específicamente para ellas. Este enfoque lo denominan los autores de `CLIP` como *zero-shot learning*, y lo utilizan para evaluar la capacidad del modelo para generalizar a conjuntos de datos no vistos previamente, lo que constituye una métrica fundamental para medir su habilidad de aprendizaje sobre tareas nuevas.

En cuanto a la arquitectura del *Image Encoder*, `CLIP` explora dos variantes. La primera emplea `ResNet-50` (He et al., 2016) como base, dada su adopción generalizada y rendimiento probado. La segunda se basa en la arquitectura *Vision Transformer* (ViT) (véase la Sección 3.5.2 dónde se explica un poco cómo funciona), al que se le realiza una pequeña modificación: se añade una capa de normalización adicional antes del paso por el transformador, y se utiliza un esquema de inicialización ligeramente distinto.

Durante el proceso de entrenamiento, se entrenaron cinco modelos basados en `ResNet` y tres basados en *Vision Transformers*. En el caso de las `ResNets`, se entrenaron un `ResNet-50`, un `ResNet-101` y tres modelos escalados siguiendo el estilo de *EfficientNet*, que utilizan aproximadamente 4x, 16x y 64x el cómputo de un `ResNet-50`, denominados respectivamente como `RN50x4`, `RN50x16` y `RN50x64`. Para los *Vision Transformers*, se entrenaron un `ViT-B/32`, un `ViT-B/16` y un `ViT-L/14`

donde la B, L hace referencia al tamaño del modelo (*base* vs *large*) y el número representa la dimensión del parche en el ViT (por ejemplo 32×32 píxeles). En este último caso, además, se realizó un preentrenamiento adicional a una resolución mayor de 336 píxeles durante una época extra para mejorar su rendimiento y le denotaron como `ViT-L/14@336px`.

En la Tabla 4.3 se detallan las resoluciones utilizadas, junto con otros datos relevantes de los modelos entrenados como la dimensión del vector de características que devuelve el codificador de imágenes (*embedding dimension*, en inglés).

Model	Dimensión Embedding	Resolución imágenes entrada
RN50	1024	224×224
RN101	512	224×224
RN50x4	640	288×288
RN50x16	768	384×384
RN50x64	1024	448×448
ViT-B/32	512	224×224
ViT-B/16	512	224×224
ViT-L/14	768	224×224
ViT-L/14-336px	768	336×336

Tabla 4.3: Hiperparámetros de `CLIP-ViT` y `CLIP-ResNet` extraídos de (Radford et al., 2021)

Como se muestra en el artículo original (Radford et al., 2021) y en la Figura 4.9 (correspondiente a la Figura 10 del citado trabajo), los modelos *CLIP basados en la arquitectura Vision Transformer (ViT)* se destacan como los más eficaces en tareas de *representation learning* dentro de su correspondiente escala computacional. En dicha figura, el eje X indica el coste computacional de inferencia (medido en GFLOPs por imagen), mientras que el eje Y representa la calidad media de las representaciones generadas por cada modelo en distintos conjuntos de datos. Esta calidad se evalúa mediante la precisión obtenida por un *clasificador lineal* entrenado sobre las características congeladas del modelo para cada conjunto de datos.

Este tipo de evaluación, basada en el desempeño de clasificadores lineales sobre representaciones congeladas, es ampliamente reconocida como una métrica sólida para valorar la calidad y generalidad de las representaciones aprendidas.

En ambas gráficas, una correspondiente a 12 conjuntos de datos extraídos de (Kornblith et al., 2019), y otra basada en 27 conjuntos de datos más diversos (los 12 anteriores más otros 15 añadidos) presentados en el apéndice del artículo original (Radford et al., 2021), se observan las siguientes tendencias destacadas:

- Los modelos `CLIP-ViT` (representados mediante estrellas rojas) dominan la región superior izquierda del gráfico, lo que refleja un excelente compromiso entre precisión y coste computacional.

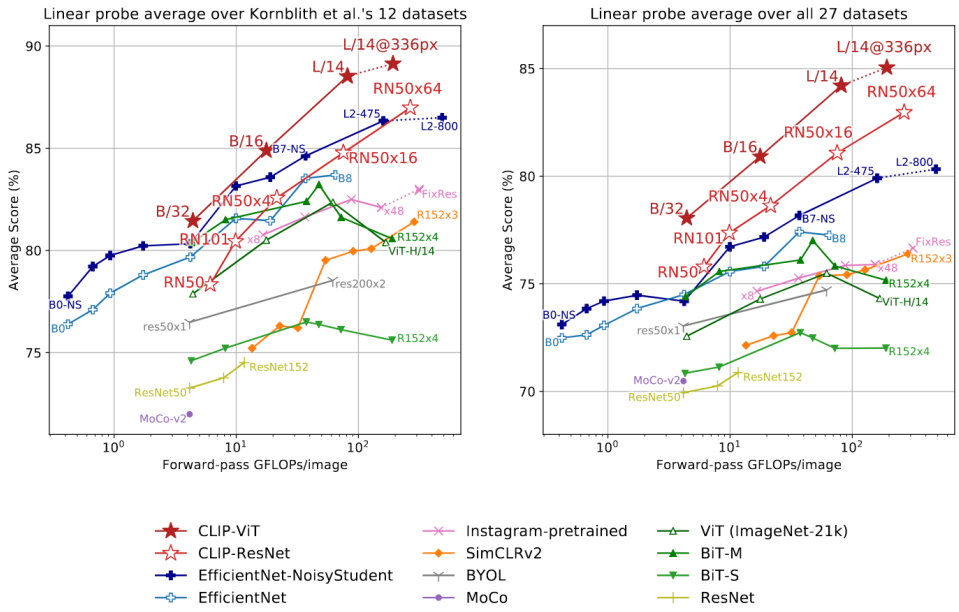

Figura 4.9: Desempeño de los modelos CLIP usando clasificadores lineales, comparado con los modelos de visión por computador más destacados del estado del arte (Figura 10 del trabajo (Radford et al., 2021)))

- En particular, el modelo `CLIP ViT-L/14@336px` alcanza el mejor rendimiento absoluto, superando incluso al modelo existente con el mejor rendimiento (`EfficientNet-L2 Noisy Student`) tanto en puntuación general como en eficiencia computacional.
- Los modelos CLIP basados en la arquitectura Vision Transformer también demuestran una escalabilidad superior respecto a los modelos basados en ResNet, permitiendo alcanzar mayores niveles de precisión con un coste computacional menor.

Dado que el objetivo de este trabajo es la geolocalización de imágenes, resulta especialmente relevante emplear un modelo con alta capacidad de representación en múltiples dominios visuales. En este sentido, la evaluación realizada sobre un conjunto de 27 *datasets* variados es de gran interés, ya que pone a prueba la capacidad del modelo para generalizar más allá de tareas tradicionales como la clasificación en ImageNet.

Entre esos 27 *datasets* utilizados para construir la Figura 4.9, destacan algunos que resultan particularmente relevantes para tareas relacionadas con la geolocalización, tales como:

- *EuroSAT* (Helber et al., 2019) : consiste en imágenes satelitales clasificadas por tipo de uso del suelo (agrícola, urbano, forestal, etc.), lo cual es directamente relevante para la geolocalización, ya que proporciona información visual geográfica explícita.
- *NWPU-RESISC45* (Cheng et al., 2017): otro *dataset* de clasificación de escenas de imágenes satelitales remotas con 45 clases distintas que incluyen áreas como

puentes, areas comerciales, autopistas, ríos, desiertos, etc. Es especialmente útil para aprender a diferenciar tipos de paisajes y estructuras geográficas.

- *KITTI* (Geiger et al., 2012): contiene imágenes tomadas desde vehículos en movimiento, en entornos urbanos, rurales y en autopistas, y se utiliza para tareas de percepción en conducción autónoma. Estas imágenes incluyen estructuras de calles, señales, edificios, etc., que son indicativas de localización.

- *Country211* (Radford et al., 2021): *dataset* diseñado para evaluar la capacidad de geolocalización de las representaciones visuales. Filtrando el conjunto de datos `YFCC100m` (Thomee et al., 2016) para encontrar 211 países (definidos como aquellos que tienen un código de país `ISO-3166` (Wikipedia, 2025b)) que tuvieran al menos 300 fotos con coordenadas GPS, y construyendo un conjunto de datos equilibrado con 211 categorías, seleccionando 200 fotos para entrenamiento y 100 fotos para prueba para cada país.

La capacidad del modelo `CLIP-ViT` para generar, en promedio, representaciones de alta calidad en una amplia gama de dominios sugiere que el espacio latente que produce es amplio, diverso y rico en características visuales. Esta propiedad es crucial para la tarea de geolocalización, ya que mi objetivo es que cada imagen representativa de una zona a geolocalizar se transforme en un vector que capture la mayor cantidad de información geográfica y contextual posible.

Al utilizar estas representaciones como entrada para entrenar un modelo de regresión destinado a predecir coordenadas de latitud y longitud, se maximiza la probabilidad de que el modelo aprenda patrones geográficos complejos. Esto se logra gracias a la riqueza semántica del espacio vectorial de representación en el que se apoyan las imágenes, lo que permite capturar relaciones visuales y geográficas profundas.

Por todo lo anterior, hemos optado por utilizar el modelo `CLIP-ViT`, pero exclusivamente la parte del codificador de imágenes, dado que solo disponemos de imágenes sin texto en nuestro conjunto de datos. El *Image Encoder* de `CLIP-ViT` es el componente adecuado para realizar la traducción de imágenes a un vector de características compacto y de alta diversidad, tal como se requiere en nuestro caso.

Este modelo presenta dos ventajas clave. Por un lado, su eficiencia computacional permite aprovechar mejor los recursos disponibles y obtener un mayor rendimiento con el mismo coste, como reflejan las tendencias vistas sobre la Figura 4.9. Por otro lado, la capacidad del ViT para modelar relaciones de largo alcanze entre parches resulta especialmente útil en tareas complejas como la geolocalización de imágenes, donde las pistas visuales geográficas relevantes pueden estar dispersas por toda la imagen, no concentrarse en una única región y necesitar de sus relaciones para conocer con mayor precisión la ubicación.

Existen varios modelos dentro de la familia `CLIP-ViT`, cuyas características se resumen en la Tabla 4.4. Una de las diferencias clave entre ellos es el *tamaño del patch*, que influye directamente en la cantidad de subdivisiones que se realizan sobre una imagen de entrada de 224×224 píxeles. Cuanto menor es el tamaño del patch, mayor es el número de segmentos en que se divide la imagen, lo que se traduce en

más *tokens* y, por tanto, en una mayor capacidad para computar relaciones entre distintas regiones de la imagen.

Modelo	Patch size	Modelo size	# tokens img	Precisión ↑	Velocidad ↓	RAM/VRAM ↓	Capas[1]	Dim[2]	Cabezas[3]
ViT-B/32	32×32	Base (B)	49	Baja	Alta	Baja	12	768	12
ViT-B/16	16×16	Base (B)	197	Media	Media	Media	12	768	12
ViT-L/14	14×14	Large (L)	257	Alta	Baja	Alta	24	1024	16
ViT-L/14-336px	14×14	Large (L)	576	Muy Alta	Muy Baja	Muy Alta	24	1024	16

Tabla 4.4: Comparativa entre distintas variantes de `ViT` en función de su tamaño de patch, arquitectura y eficiencia.

Por ejemplo, utilizando un tamaño de patch de 32×32 píxeles, el número total de *tokens* resulta de $\frac{224 \times 224}{32 \times 32} = 7 \times 7 = 49$ *tokens*. Si el tamaño del patch disminuye, este número aumenta, enriqueciendo el conjunto de representaciones disponibles para el modelo.

Además del tamaño del patch, otra dimensión fundamental es el tamaño del modelo: `ViT-B` (Base) cuenta con una arquitectura de 12 capas de transformador, una dimensionalidad interna de 768 y 12 cabezas de atención por capa. Por su parte, `ViT-L` (Large) incrementa significativamente estas capacidades, con 24 capas, una dimensionalidad de 1024 y 16 cabezas de atención, lo que le permite modelar relaciones más complejas entre los *tokens*.

En resumen, existen dos factores esenciales a tener en cuenta: (i) *el tamaño del patch*, donde a menor tamaño, mayor número de *tokens*, lo que permite aumentar el potencial de capturar relaciones más detalladas entre distintas regiones de la imagen, (ii) *el tamaño del modelo*, donde arquitecturas más grandes (mayor número de capas, dimensionalidad y cabezas de atención) proporcionan una mayor capacidad para modelar esas relaciones de forma efectiva, siempre que se cuente con recursos suficientes para su entrenamiento.

Considerando que necesitamos generar los *embeddings*, vector de características, para más de 5 millones de imágenes y tomando en cuenta las limitaciones computacionales y de tiempo disponibles, se decide optar por los modelos `ViT-B/32` y `ViT-B/16` (respectivamente el más ligero y rápido; y el de mejor compromiso entre velocidad y precisión). De acuerdo con los tiempos mostrados en la Tabla 4.5, estos modelos ofrecen un equilibrio entre gasto computacional y calidad de representación, como se evidenció en la Figura 4.9 y en la Tabla 4.4. Aunque el modelo `ViT-L/14@336px` ofrece el mejor rendimiento absoluto en términos de precisión, su coste computacional es considerablemente mayor, lo que nos lleva a descartarlo, junto con `ViT-L/14` que, aunque no tanto, es también bastante costoso, y elegir los otros dos modelos. En consecuencia, desarrollaremos dos familias de submodelos entrenados con conjuntos de imágenes codificadas con estos *Image Encoder*, ya que, dentro de sus limitaciones computacionales, son los que ofrecen los mejores resultados.

[1]Referidas a las capas del transformador
[2]Referidas a la dimensión interna en la representación de atención
[3]Referidas a las cabezas de atención en cada capa

Modelo	T. embeddings 50000 imágenes	T. Total
ViT-B/32	18min 18,2s	29h 51min 44,9s
ViT-B/16	71min 22,9s	116h 27min 46,4s
ViT-L/14	5h 53min 54s	≈ 577h 24min 34,4s
ViT-L/14-336px	11h 37min 29,53s	≈ 1138h 4,5s

Tabla 4.5: Tiempos Modelos CLIP-ViT sacando embeddings de imágenes

4.3.3. Módulo de Geolocalización

El Módulo de Geolocalización constituye la segunda etapa fundamental del modelo propuesto, encargado de transformar la representación vectorial generada por el codificador de imágenes y posteriormente normalizada usando normalización L_2, en una predicción precisa de coordenadas geográficas. Para ello, se ha optado por una arquitectura basada en un Perceptrón Multicapa (ver Sección 3.1) debido a su capacidad para aproximar funciones complejas.

La elección de un MLP se fundamenta en dos motivos principales:

- Los perceptrones multicapa, con su conectividad densa y la profundidad y tamaño de sus capas intermedias, son de las mejores arquitecturas para la geolocalización de imágenes. Estas características permiten al modelo construir representaciones progresivamente más abstractas del rico espacio vectorial proporcionado por el *Image Encoder*, lo cual es crucial en un problema tan complejo que depende de la combinación de múltiples características visuales.

- La capacidad de modelar relaciones no lineales y complejas es esencial en este contexto donde la relación entre píxeles y coordenadas geográficas es altamente no lineal. El modelo debe ser capaz de interpretar una amplia variedad de claves visuales, como vegetación, arquitectura o clima, que no mantienen una correlación lineal con la latitud y la longitud. Y es que una relación lineal implica que pequeños cambios constantes en la entrada supongan pequeños cambios constantes en la salida, sin embargo, pequeñas variaciones en una imagen pueden corresponder a grandes diferencias geográficas, y viceversa. Un ejemplo ilustrativo de esta complejidad se muestra en la Figura 4.10. En ella, la mayoría de las personas puede reconocer que la imagen de la izquierda corresponde a la Torre Eiffel, ubicada en París, Francia. Sin embargo, un experto es capaz de identificar que la imagen de la derecha muestra una réplica de dicha torre, localizada en Shenzhen, China. Esta distinción no se basa en la estructura principal, que es visualmente similar en ambas imágenes (lo que haría que potencialmente se encontrasen relativamente cerca en el espacio de características), sino en pistas sutiles del entorno, como la presencia de otros monumentos replicados o el estilo de los edificios del fondo. Así, dos imágenes muy parecidas visualmente pueden estar separadas por miles de kilómetros (más de 9000 km), reforzando la necesidad de que el modelo sea capaz de capturar relaciones no lineales de manera efectiva.


Figura 4.10: Similitud de Imágenes Distantes (Izbicki et al., 2020)

4.3.3.1. Arquitectura del Módulo de Geolocalización

La arquitectura del `MLP` queda determinada por el número de capas, la cantidad de neuronas en cada una de ellas y las funciones de activación utilizadas en las distintas capas.

Todos los modelos comparten una estructura base común: $d \to 256 \xrightarrow{\text{ReLU}} 128 \xrightarrow{\text{ReLU}} 2$. La dimensión d en la capa de entrada viene dada por el vector de características extraído por el *Image Encoder* (que en nuestro caso va a ser 512), y la capa de salida tiene tamaño 2 porque el objetivo del modelo es predecir dos variables continuas: latitud y longitud.

La decisión de tener dos capas intermedias y con esa cantidad de neuronas, 256 y 128 respectivamente, responde a una intuición de sintetización progresiva de la información. Partiendo del vector de características de dimensión d (512), se busca ir reduciendo gradualmente las neuronas a la mitad ($d \mapsto 256 \mapsto 128...$) para que esas características potencialmente se vayan combinando y transformando en representaciones más compactas y relevantes, de modo que en la última transición ($...128 \mapsto 2$) se pueda estimar de forma precisa la ubicación geográfica a partir de esa información ya condensada y fusionada. No se añaden más capas para evitar un aumento innecesario del tiempo de entrenamiento, buscando un equilibrio entre precisión y eficiencia computacional.

El uso de la función de activación ReLU para salida de las capas ocultas se debe a que es una de las opciones más habituales en redes neuronales. Esto se debe, como ya se había dicho en la Sección 3.1.1 a tres razones:

- Introduce no linealidad, lo cual es crucial para modelar relaciones complejas y patrones no lineales entre los datos de entrada y salida
- Su simplicidad en el cálculo: la fórmula $\text{ReLU}(x) = \text{máx}(0, x)$ la hace computacionalmente eficiente. Además, su derivabilidad sencilla facilita el cálculo de gradientes durante el proceso de retropropagación.
- La capacidad de transformar cualquier valor negativo en 0 (no aproximado, sino exactamente 0) permite la generación de activaciones dispersas (*sparse activations*, en inglés) (Awasthi et al., 2024). Esto significa que solo una parte de las neuronas se activan en cada capa, lo que puede aumentar la eficiencia

computacional de la red, reducir la propensidad al sobreajuste y favorecer el aprendizaje de representaciones más específicas. Esto se puede comparar con la idea de que diferentes partes de la red se especialicen en detectar aspectos como el idioma, la flora, etc. Si alguna de estas características no se detecta en el vector de dimensión d, esas neuronas no participarán en la geolocalización de la imagen correspondiente.

Partiendo de esta arquitectura base, se han propuesto varias modificaciones que se aplicarán a la capa de salida para crear varios modelos que serán entrenados, cada uno con una de estas alternativas, en busca de observar cuál de ellos ofrece los mejores resultados. Estas modificaciones se han hecho en la capa de salida para aplicar una normalización simple de escalado para asegurar que la red neuronal siempre devuelva un valor de coordenada válido:

- *Postprocesamiento con* `clamp`: Se utiliza la función `clamp` de `PyTorch` (PyTorch, 2025) para ajustar la salida de la última capa de modo que los valores se encuentren dentro de un rango específico. La latitud se limita entre $[-90, 90]$ y la longitud entre $[-180, 180]$. Este ajuste se realiza después de que el modelo ha generado su salida, imponiendo un valor mínimo y/o máximo para garantizar que los valores fuera de este rango se ajusten a los límites establecidos.

- *Función de activación* `tanh` (ver Sección 3.1.1): Se añade la función de activación `tanh` en la capa de salida para que los valores producidos estén restringidos al intervalo $(-1, 1)$. Luego, se escala manualmente este intervalo para mapearlo a las coordenadas válidas de latitud y longitud ($[-90, 90]$ y $[-180, 180]$, respectivamente). Este enfoque busca permitir que la red neuronal se enfoque en aprender valores dentro de un rango más reducido, lo que podría para este problema en particular facilitar la predicción de coordenadas válidas tras el escalado.

- *Función de activación logística* (ver Sección 3.1.1): Se aplica la función de activación logística a la capa de salida para que los valores estén entre 0 y 1, y luego se realiza un postprocesamiento manual para escalar estos valores a los rangos válidos de latitud y longitud. Esta estrategia, similar a la anterior, busca reducir aún más el rango de valores que la red neuronal debe aprender a generar, creyendo que quizá de esta manera se facilita el proceso de aprendizaje.

La motivación de estas modificaciones radica en sus enfoques complementarios. La primera opción es la más directa e intuitiva: consiste en aplicar a la salida de la red neuronal, que inicialmente no está limitada, una restricción posterior actuando como una barrera para evitar coordenadas fuera de rango. En cambio, las otras dos modificaciones se idearon para restringir desde el inicio el rango de valores en la salida posibles, entrenando la red con esa limitación en mente y luego escalar adecuadamente el resultado.

4.3.4. Parámetros de Entrenamiento

En esta sección, se detallarán los valores seleccionados para los hiperparámetros de entrenamiento (ver Sección 3.4.2) utilizados en los modelos desarrollados en este proyecto. Estas configuraciones son clave para determinar cómo se desarrolla el entrenamiento, el uso de recursos computacionales, la duración del proceso y el rendimiento final del modelo.

En cuanto al parámetro de paciencia introducido en la Sección 3.4.2.1, el valor seleccionado es de 5 épocas, ya que hemos observado que el error en el conjunto de validación mejora rápidamente al principio, pero luego tiende a estancarse, por lo que queremos evitar que el error en el conjunto de validación se desvíe demasiado del error en el conjunto de entrenamiento.

En el caso de la tasa de aprendizaje (ver Sección 3.4.2.2), el valor seleccionado es 0,001, que es el valor por defecto en `PyTorch` (PyTorch, 2025) para el optimizador `Adam`, que se detallará más adelante. Aunque puede parecer un valor estático, en realidad se trata de una tasa base, ya que `Adam` aplica una tasa de aprendizaje efectiva que varía automáticamente.

Barajando entre las alternativas diponibles de *batch size* presentadas en la Sección 3.4.2.3, la primera opción es inviable, ya que el conjunto de datos de entrenamiento, con más de 4 millones de datos, no cabe en la memoria de la GPU de la máquina de entrenamiento, que tiene 11,9 GB de memoria. Por lo tanto, entre las otras dos opciones se opta por la tercera opción (*Mini-Batch Gradient Descent*), que es la más equilibrada, permitiendo que el modelo pueda caber en la GPU y ofreciendo un balance entre el ruido del `SGD`, que ayuda a escapar de los mínimos locales, y la estabilidad del *Batch Gradient Descent*. Para evitar un consumo energético y de hardware excesivo y teniendo en cuenta las limitaciones del ordenador, se ha decidido usar un tamaño de lote de 64 que representa un compromiso entre un tamaño de lote alto y bajo.

4.3.5. Funciones de Pérdida y Métricas

En esta sección se van a definir las distintas funciones de pérdida (ver Sección 3.4.3) empleadas en el entrenamiento de los diferentes modelos desarrollados en el proyecto.

Como se explicó en el marco teórico, hay dos tipos de funciones de pérdida según la naturaleza del problema (regresión vs clasificación). En el caso que nos ocupa, la geolocalización mediante la predicción de unas coordenadas de latitud y longitud, estamos claramente ante un problema de regresión, ya que las salidas del modelo son valores numéricos continuos que representan posiciones geográficas. Por tanto, es necesario que todos los modelos empleen funciones de pérdida propias de regresión, que se ajusten a las características y necesidades de este tipo de tarea.

La elección de la función de error es crucial puesto que se pueden estar priorizando unos errores frente a otros según la función que se esté usando. Por eso, la función de pérdida adquiere un papel especialmente relevante, ya que puede influir directamente en el tipo de aprendizaje que realiza el modelo. Por ejemplo, puede diseñarse para penalizar fuertemente predicciones muy alejadas de la ubicación real, o bien para

recompensar con mayor énfasis aquellas cercanas, ignorando errores grandes pero infrecuentes. Por esta razón, se han desarrollado múltiples modelos entrenados con distintas funciones de pérdida, cada una diseñada con un enfoque específico que será detallado a continuación.

4.3.5.1. Primera aproximación: MSELoss

Comencé utilizando la función de pérdida más común y extendida en tareas de regresión: el *Error Cuadrático Medio* (*Mean Squared Error*, MSE). Esta función es continuamente diferenciable, lo cual la hace especialmente adecuada para la optimización de modelos mediante algoritmos basados en gradiente. Además, su implementación está incluida de forma nativa en `PyTorch`, lo que permite integrarla fácilmente sin necesidad de programarla desde cero.

Tal como sugiere su nombre, el MSE se calcula como la media de los errores cuadráticos entre los valores predichos y los reales para cada ejemplo del lote. La fórmula en (PyTorch, 2025) se define a partir del *error cuadrático* que para una salida de dimensión k se expresa como:

$$\text{SquaredError} = \frac{1}{k}\sum_{i=1}^{k}(\hat{y}_i - y_i)^2 \tag{4.2}$$

Donde:

- k es la dimensión del vector de salida.
- $\hat{y}_i$ es la componente i-ésima del vector de salida predicho.
- y_i es la componente i-ésima del vector de salida real.

A partir de este error cuadrático, la pérdida total del lote se obtiene como la media sobre todos los ejemplos:

$$\text{MSELoss} = \frac{1}{N}\sum_{j=1}^{N}\text{SquaredError}_j$$

Donde:

- N es el tamaño del lote (*batch size*).
- SquaredError_j corresponde al error cuadrático definido en la Ecuación 4.2 para el ejemplo j-ésimo del lote.

En el caso de este trabajo $k = 2$ ya que hace referencia a las coordenadas de latitud y longitud que tiene que predecir el modelo de regresión.

El hecho de elevar al cuadrado el error implica que la pérdida siempre será positiva, evaluando únicamente la magnitud del error, sin considerar su dirección. Además, al aplicar el cuadrado, los errores grandes tienen un impacto desproporcionadamente alto sobre la pérdida total. Esto penaliza fuertemente los valores atípicos e incentiva al modelo a reducirlos activamente.

No obstante, esta función no es perfecta. Su aplicabilidad se restringe a espacios vectoriales en $\mathbb{R}^k$, y no considera la naturaleza esférica de la superficie terrestre. Por esta razón, errores similares en magnitud pueden tener implicaciones muy diferentes dependiendo de la región geográfica. Por ejemplo, errores cercanos entre sí pueden ser más tolerables ya que al tener la superficie esférica un radio tan grande, distancias cercanas actúan como si estuviesen en el plano, mientras que errores de mayor magnitud pueden resultar mucho más significativos y nada representativos. A pesar de estas limitaciones, el uso de MSE resulta útil como línea base funcional sobre la cual construir y experimentar para tareas de geolocalización.

4.3.5.2. Mejora con métrica geográfica: HaversineLoss

Medir coordenadas geográficas utilizando la pérdida MSE es conceptualmente similar a intentar medir distancias sobre la superficie de una esfera utilizando una regla recta: no es una aproximación precisa ni adecuada. El problema fundamental radica en que el MSE asume un espacio euclídeo, ignorando completamente la curvatura de la superficie terrestre. Para abordar esta limitación geométrica se recurre a una alternativa más adecuada: la *distancia Haversine* (Brummelen, 2013).

La distancia Haversine calcula la distancia del círculo máximo entre dos puntos sobre la superficie de una esfera, siendo esta la ruta más corta sobre dicha superficie dada la latitud y longitud de ambos puntos. Esta métrica es ampliamente utilizada en navegación y geolocalización, ya que proporciona una estimación más realista de la separación entre ubicaciones geográficas. No considera accidentes geográficos como montañas o valles, asumiendo que la superficie terrestre es perfectamente esférica y que la distancia se mide "en línea recta" a través del aire o del terreno idealizado.

La fórmula es la siguiente:

$$\begin{aligned} a &= \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cdot\cos(\varphi_2)\cdot\sin^2\left(\frac{\Delta\lambda}{2}\right) \\ c &= 2\cdot\operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right) \\ d &= R\cdot c \end{aligned} \tag{4.3}$$

Donde:

- φ_1, φ_2 son las latitudes de los dos puntos (en radianes).
- λ_1, λ_2 son las longitudes de los dos puntos (en radianes).
- $\Delta\varphi = \varphi_2 - \varphi_1$ es la diferencia de latitud.
- $\Delta\lambda = \lambda_2 - \lambda_1$ es la diferencia de longitud.
- R es el radio de la Tierra, típicamente estimado en 6370 km.

El valor c representa la distancia angular (en radianes) entre los dos puntos, es decir, el ángulo que describiría un arco trazado entre ellos sobre la circunferencia máxima de la esfera. Finalmente, multiplicando esta distancia angular por el radio

terrestre, se obtiene la distancia real d sobre la superficie de la Tierra nunca pudiendo ser menor que 0.

Un aspecto importante de esta formulación es que la distancia de Haversine es una función diferenciable y por tanto adecuada para optimizaciones basadas en gradientes. Esto se debe a que está compuesta por funciones trigonométricas diferenciables, como el seno y el coseno, así como por la raíz cuadrada, la cual es diferenciable en su dominio positivo. Además, la función `atan2` es diferenciable en todo su dominio excepto en el punto $\text{atan2}(0, 0)$, el cual no se alcanza en este caso, ya que $\sqrt{a}$ y $\sqrt{1-a}$ no pueden ser simultáneamente cero.

La expresión del término intermedio a garantiza que $a \geq 0$, ya que:

- Los términos $\sin^2\left(\frac{\Delta\varphi}{2}\right)$ y $\sin^2\left(\frac{\Delta\lambda}{2}\right)$ están acotados entre 0 y 1.
- Las funciones $\cos(\varphi_1)$ y $\cos(\varphi_2)$ toman valores en $[0, 1]$ cuando $\varphi_1, \varphi_2 \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, como ocurre con las latitudes terrestres en radianes.

Además, $a \leq 1$ ya que en el peor caso, cuando $\sin^2\left(\frac{\Delta\lambda}{2}\right) = 1$, la expresión sigue acotada por ese valor. Usando la identidad del ángulo mitad:

$$\sin\left(\frac{\theta}{2}\right) = \pm\sqrt{\frac{1-\cos(\theta)}{2}}$$

Podemos escribir una cota superior:

$$\begin{aligned} a &\leq \frac{1-\cos(\varphi_2-\varphi_1)}{2} + \cos(\varphi_1)\cdot\cos(\varphi_2) \\ &= \frac{1-\cos(\varphi_2-\varphi_1)+2\cos(\varphi_1)\cos(\varphi_2)}{2} \end{aligned}$$

Desarrollando $\cos(\varphi_2-\varphi_1)$ con la fórmula del coseno de la diferencia de ángulos, reorganizando y aplicando la fórmula del coseno de la suma de ángulos, obtenemos:

$$a \leq \frac{1-\cos(\varphi_1+\varphi_2)}{2} = \cos^2\left(\frac{\varphi_1+\varphi_2}{2}\right)$$

En la última igualdad se ha aplicado el cuadrado de la identidad del ángulo mitad. Así garantizamos que $a \leq 1$, y concluimos $a \in [0, 1]$ asegurando que la raíz cuadrada esté bien definida y que la función completa sea diferenciable.

Cabe señalar que la distancia calculada mediante la fórmula 4.3 corresponde a un único par de coordenadas (predicción y valor real). En el contexto del entrenamiento de modelos, la función de pérdida basada en la distancia de Haversine se aplica sobre un lote de tamaño N, calculando la distancia para cada una de las N predicciones con respecto a sus ubicaciones reales, y promediando el resultado:

$$\text{HaversineLoss} = \frac{1}{N}\sum_{i=1}^{N} d_i$$

donde d_i es la distancia Haversine del ejemplo i entre la predicción y la ubicación real.

No obstante, esta métrica también presenta ciertas limitaciones: al centrarse exclusivamente en la distancia física entre dos puntos, no ofrece flexibilidad para introducir penalizaciones o bonificaciones adicionales que podrían resultar relevantes durante el entrenamiento. Por ejemplo, podría ser deseable penalizar más fuertemente errores especialmente grandes o, por el contrario, recompensar con mayor intensidad las predicciones muy cercanas al valor real.

A partir de aquí, todas las referencias a la distancia entre dos puntos en la Tierra asumirán, salvo que se indique lo contrario, el uso de la distancia Haversine, ya que todas las métricas posteriores se construyen sobre esta base.

4.3.5.3. Inspiración práctica: GeoscoreLoss

Además de la distancia Haversine, uno puede inspirarse en el juego *GeoGuessr* (GeoGuessr, 2025) y observar su forma de puntuar la precisión de sus jugadores. A diferencia de una penalización lineal, *GeoGuessr* aplica una caída exponencial en los puntos: cuanto más cerca se está del objetivo, mayor es el impacto de la distancia en la puntuación. Por ejemplo, estar a 30 km de distancia en un mapa mundial puede suponer una pérdida de unos 100 puntos, mientras que la diferencia entre estar a 1000 km y a 1030 km es inferior a 50 puntos. Además, este sistema denominado *GeoScore*, recompensa con una puntuación cercana a 5000, el valor máximo, las predicciones muy precisas (por ejemplo, entre 3 y 10 km de error), pero deja de penalizar más allá de cierto umbral, asignando directamente una puntuación de 0. Aunque la fórmula exacta no es pública, la comunidad ha logrado aproximar su comportamiento, como se muestra en la guía de referencia (Plonkit Team, 2025). La función estimada es la siguiente:

$$5000 \cdot e^{-10 \cdot \left(\frac{\text{Distance}}{\text{Max Distance}}\right)} \tag{4.4}$$

Donde:

- Distance representa la distancia Haversine entre la predicción y la ubicación real.
- Max Distance es la diagonal del mayor rectángulo que engloba cualquier posible ubicación en el mapa.

Esta fórmula refleja una disminución no lineal en la puntuación, acercándose asintóticamente a 0 para grandes distancias, lo que introduce el umbral implícito que se comentaba.

Llamaremos *factor de escala* a la razón $\frac{\text{Distance}}{\text{Max Distance}}$, que normaliza la distancia respecto a la máxima posible en el mapa. Cuando esta proporción se aproxima a 1, es decir, cuando Distance $\approx$ Max Distance, la puntuación se reduce a aproximadamente $5000 \cdot e^{-10} \approx 0{,}22$, prácticamente cero.

El valor de Max Distance depende del mapa, ya que *GeoGuessr* permite crear mapas personalizados que engloban algunas zonas del mundo. En este trabajo, sin

embargo, se asume un mapa global. Si consideramos la Tierra como una esfera, la distancia máxima entre dos puntos en su superficie sería la mitad del perímetro del ecuador, es decir, $40075/2 = 20037{,}5$ km.

Dado que se quiere definir una función de pérdida para entrenamiento supervisado, esta debe ser creciente respecto a la distancia (a mayor error, mayor pérdida). Como la fórmula original 4.4 es decreciente y su mayor puntuación alcanzable es 5000, se realiza la siguiente transformación para adaptarla:

$$5000 - 5000 \cdot e^{-10 \cdot \left(\frac{\text{Distance}}{20037{,}5}\right)}$$

Denominaremos esta función como *GeoscoreLoss*, ya que se inspira directamente en el *GeoScore* de *GeoGuessr*. Esta función es diferenciable, al componerse de una función exponencial (diferenciable) y de la distancia Haversine (también diferenciable). Como en el caso de la pérdida basada únicamente en Haversine, esta pérdida se calcula para cada ejemplo individual y luego se promedia en el lote de entrenamiento.

La función *GeoscoreLoss* refleja de forma más fiel la percepción humana sobre la calidad de una predicción geográfica. Como puede observarse en la Figura 4.11, las predicciones muy cercanas al objetivo reciben valores de pérdida extremadamente bajos, premiando con fuerza la alta precisión. A medida que la distancia aumenta, la penalización crece rápidamente, reflejando una caída abrupta en la valoración de la predicción. Sin embargo, este crecimiento se va suavizando progresivamente hasta estabilizarse en una pérdida máxima de 5000. Este comportamiento resulta especialmente valioso: al presentar una pendiente inicial muy pronunciada, la función de pérdida incentiva de manera fuerte las predicciones exactas, algo que es especialmente deseable en tareas como la geolocalización, donde estar "muy cerca" es lo más deseable. Por otro lado, una vez que el error supera cierto umbral y la predicción ya se considera fallida, seguir penalizando de forma creciente puede llegar a perder el sentido, y por eso la función se aplana.

No obstante, este comportamiento asintótico para distancias grandes puede representar una limitación durante el entrenamiento. Al estabilizarse la pérdida en $y = 5000$ para grandes errores, el gradiente tiende a desaparecer, dificultando la corrección de las predicciones muy alejadas de la solución real.

4.3.5.4. Intervención manual: LinealScoreLoss

No satisfecho con los enfoques anteriores, decidí intervenir directamente y definir cómo se penalizarían las predicciones en función de la distancia. Para ello, introducimos dos conceptos clave:

- *Tamaño "ciudad"* (TC): distancia máxima que aún se consideraría aceptable para otorgar una muy buena puntuación, como si ambos puntos se encontrasen en la misma "ciudad".

- *Diámetro de la mitad de un país* (DMP): la mitad de la distancia entre los dos puntos más alejados dentro de un país promedio.

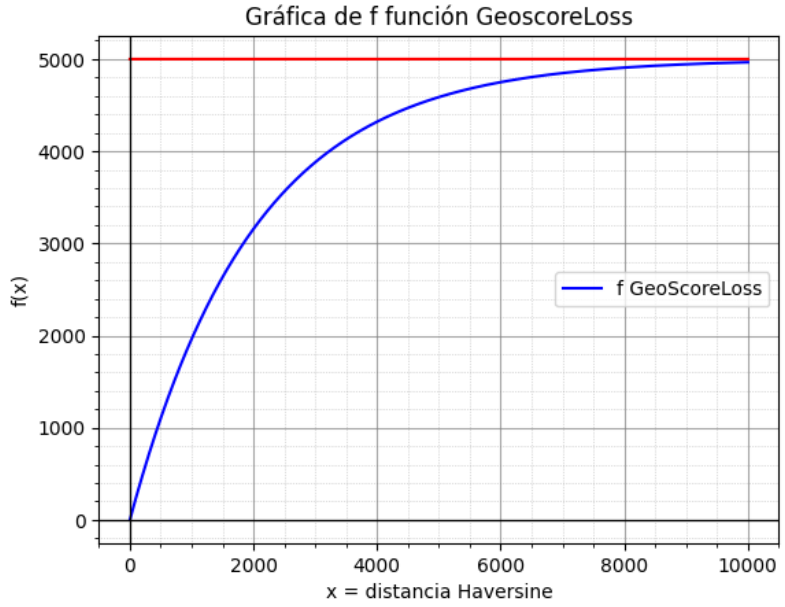

Figura 4.11: Gráfica de la función GeoscoreLoss

Considero que una distancia de error razonable para un modelo capaz de ubicar una localización a partir de una sola imagen, ronda los 30 km. Por ello, definimos $TC = 30$ km, considerando que predicciones con errores inferiores a esa distancia son respetables.

Para calcular el DMP, recurrimos a datos más precisos. Tomamos el promedio del área de 250 entidades geográficas listadas en `REST Countries` (Matos, 2025), incluyendo estados soberanos, territorios dependientes y regiones con estatus político especial. El resultado fue una media de 600 584,81 km^2. Asumiendo que los países tienen forma circular, y usando la fórmula del área de una circunferencia $A = \pi r^2$, obtenemos: $\sqrt{\frac{600584{,}81}{\pi}} \approx 437{,}23$ km. Por lo tanto, fijamos $DMP = 437{,}23$ km.

Con estos valores definidos, es posible determinar y visualizar mejor qué penalizaciones se asocian a qué distancias, más allá del valor numérico. En la Tabla 4.6 se presentan las aproximaciones de penalización:

Distancia Real	Distancia de la métrica	Descripción
0–30 km	0	Distancias buenas o aceptables (misma "ciudad")
437,23 km (DMP)	500	Mitad de un país
874,46 km ($2 \cdot$ DMP)	1000	Un país completo
1748,92 km ($4 \cdot$ DMP)	2000	Dos países
5000 km	5000	Valores extremos, equivalentes a los de *GeoScoreLoss*
$>$ 5000 km	$>$ 5000	Valores aún más extremos

Tabla 4.6: Penalizaciones en la métrica LinealScoreLoss

Esta penalización es prácticamente lineal: al duplicar el DMP, también se duplica la distancia de la métrica. Por ello, se propone una función lineal de la forma $y = mx + c$, donde x representa la distancia Haversine. Esta aproximación simplifica

el cálculo, mejora la eficiencia computacional y evita complicaciones innecesarias, ya que un ajuste más complejo no aportaría grandes beneficios con respecto lo que podría llegar a complicarlos.

Sabemos que en $x = 0$ debe cumplirse $y = 0$, ya que no tendría sentido penalizar una predicción perfecta. Esto implica que $y = m \cdot 0 + c = 0 \Rightarrow c = 0$. Además, debe cumplirse que $m > 0$, pues una pendiente negativa o nula invalidaría el propósito de penalización creciente. Aunque esta función lineal se asemeja a la distancia Haversine ($y = x$), está fundamentada en distancias relativas significativas como las mencionadas anteriormente.

Para estimar un valor adecuado de m, usamos los valores de la Tabla 4.6. Aplicando la fórmula despejada $m = \frac{y}{x}$, obtenemos:

$$m_2 = \frac{500}{437{,}23} \approx 1{,}1436, \quad m_3 = \frac{1000}{874{,}46} \approx 1{,}1436,$$
$$m_4 = \frac{2000}{1748{,}92} \approx 1{,}1436, \quad m_5 = \frac{5000}{5000} = 1$$

Promediando estas pendientes, obtenemos: $m = \frac{m_2+m_3+m_4+m_5}{4} \approx 1{,}1077$. Es importante destacar que, dado que $m > 0$, no se cumple que $f(30) = 0$, sino que: $f(30) = m \cdot 30 \approx 33{,}23 > 0$. No obstante, esta penalización es relativamente baja y no se aleja tanto del valor ideal 0 por ser una "buena distancia" así que se considera aceptable.

Al aplicar esta función de pérdida a los puntos que buscábamos aproximar, se obtienen los valores de la Tabla 4.7, los cuales se ajustan razonablemente bien a las expectativas. La Figura 4.12 muestra visualmente dicha correspondencia.

Distancia Real	**Distancia métrica**	**Distancia esperada**
0 km	0	0
30 km	33,23	0
437,23 km (DMP)	484,31	500
874,46 km ($2 \cdot$ DMP)	968,62	1000
1748,92 km ($4 \cdot$ DMP)	1937,23	2000
5000 km	5538,36	5000
$>$ 5000 km	$>$ 5538,36	$>$ 5000

Tabla 4.7: Valores obtenidos con la métrica LinealScoreLoss

Denominamos a esta función de pérdida *LinealScoreLoss* por su comportamiento lineal. Al igual que las anteriores, esta función calcula la pérdida de un único ejemplo, y durante el entrenamiento se promedia el error del lote.

Además de su simplicidad, una ventaja importante es que es claramente derivable, lo cual la hace ideal para el cálculo de gradientes. No obstante, presenta una limitación: su pendiente es muy cercana a 1, lo que la hace muy similar a la distancia Haversine, véase Figura 4.12. Aunque una pendiente ligeramente superior puede

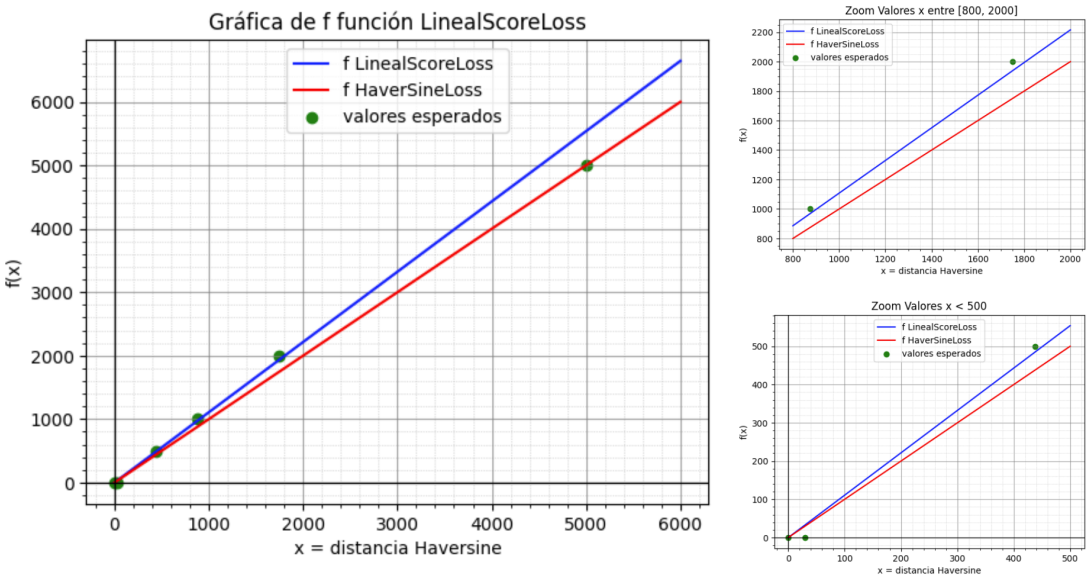

Figura 4.12: Gráfica de la función LinealScoreLoss

aportar cierta ventaja al reducir el tiempo de entrenamiento o facilitar el ajuste de pesos, su principal valor radica en ofrecer una base clara para visualizar y justificar las penalizaciones aplicadas en función de la distancia.

4.3.5.5. Enfoques Opuestos de Penalización en la Función de Pérdida

Siguiendo con la idea de desarrollar funciones de pérdida personalizadas, se presentan a continuación dos métricas adicionales nacidas de dos enfoques conceptualmente opuestos que decidí explorar por separado:

- *Enfoque de Suficiencia Tolerante:* diseñar una función que recompense bastante aquellas predicciones consideradas como "suficientemente buenas", con un crecimiento lento en la pérdida mientras el error se mantenga dentro de ese umbral de suficiencia, y un incremento agresivo a partir de ahí. El objetivo es que la red neuronal aprenda a priorizar siempre predicciones razonablemente cercanas valorando mucho el "estar suficientemente cerca". Este concepto se definirá con mayor precisión en la Sección 4.3.5.6.

- *Enfoque de Precisión Exigente:* diseñar una función que incentive especialmente las buenas predicciones, penalizando con fuerza los errores cerca del objetivo. Es decir, cuanto más próxima esté la predicción, mayor será el impacto de la penalización por el error cometido, generando así una pendiente inicial muy pronunciada. Por el otro lado, a medida que aumenta la distancia esta función buscará reducir gradualmente las penalizaciones hasta estabilizarlas, evitando castigos excesivos cuando el error sea considerable. Estas características en el gradiente buscan sacar rápidamente a la red de zonas de error leve y acelerar la convergencia hacia regiones óptimas. Esta forma de valorar los aciertos refleja

una intuición humana clara: estar "muy cerca" es mucho más valioso que estar simplemente "menos lejos". Esta filosofía recuerda a la dinámica de la métrica *GeoScoreLoss* (Sección 4.3.5.3), aunque en esta propuesta se ajustará el comportamiento final para que la estabilización no implique una pérdida constante que pueda resultar en derivadas nulas, algo poco deseable para el entrenamiento. Esta propuesta se describe con más detalle en la Sección 4.3.5.7.

Como puede observarse, ambos enfoques responden a ideas opuestas: mientras que el primero se centra en fomentar resultados "suficientemente aceptables" y establecer penalización clara sólo cuando ese umbral de suficiencia no es alcanzado, el segundo promueve la búsqueda de la "excelencia", partiendo de la premisa de que llega un punto donde estar muy cerca del resultado correcto vale mucho más que estar sólo un poco menos lejos.

4.3.5.6. Enfoque de Suficiencia Tolerante: InterCubicMonotonusHermiteLoss

Como se ha comentado previamente, esta métrica se fundamenta en la idea de recompensar aquellas predicciones consideradas como "suficientemente buenas", comenzando con un incremento de penalización suave y luego una escalada agresiva una vez se pase ese umbral de suficiencia.

La noción de lo que se entiende por "suficientemente cerca" o "suficientemente buena" se construye sobre los conceptos introducidos en la Sección 4.3.5.4, en particular el Tamaño de una "ciudad" (TC) y el Diámetro de la Mitad de un País (DMP). En este contexto, consideramos como aceptable fallar hasta aproximadamente "dos países", es decir, una distancia máxima de $4 \cdot \text{DMP} = 1748{,}92$ km. No obstante, no tratamos todas las distancias de forma uniforme: aquellas inferiores a TC recibirán una penalización nula, mientras que desde ese punto la penalización crecerá de manera suave y a partir del $4 \cdot \text{DMP}$ se volverá progresivamente más severa, alcanzando un valor elevado en los 5000 km y más aún a los 21 000 km.

Este comportamiento se define de forma explícita asignando manualmente penalizaciones concretas a distancias clave, como se muestra en la Tabla 4.8.

Índice	Distancia Real (x_i)	Distancia métrica (y_i)	Descripción
$i = 1, 2$	0–30 km	0	Distancias buenas o aceptables (misma "ciudad")
$i = 3$	100 km	50	Varias "ciudades" de separación
$i = 4$	218,615 km ($\frac{1}{2} \cdot \text{DMP}$)	100	Un cuarto de un país
$i = 5$	437,23 km (DMP)	180	Mitad de un país
$i = 6$	874,46 km ($2 \cdot \text{DMP}$)	400	Un país completo
$i = 7$	1748,92 km ($4 \cdot \text{DMP}$)	1000	Dos países
$i = 8$	5000 km	5000	Penalización extrema (como en *GeoScoreLoss*)
$i = 9$	21000 km	100000	Distancias físicamente imposibles en la Tierra[4]

Tabla 4.8: Penalizaciones en la métrica InterCubicMonotonusHermiteLoss

[4]Distancia imposible porque vimos en la Sección 4.3.5.3, la mayor distancia posible en la Tierra es de 20 037,5 km.

Para interpolar estos puntos, se emplea una Interpolación Cúbica Monótona de Hermite, un tipo de spline que garantiza una función continua, derivable y monótonamente creciente, propiedades deseables para una función de pérdida utilizada en el entrenamiento de modelos. Este enfoque asegura que los valores especificados en la tabla sean respetados exactamente, y que la función tenga derivadas continuas en todo su dominio. Para su implementación, se siguen los pasos descritos en (Knott, 1999; Wikipedia, 2025a), comenzando por calcular las pendientes de las rectas secantes entre pares de puntos consecutivos:

$$\delta_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \quad \forall i \in \{1, \ldots, 8\}$$

y las tangentes en cada punto:

$$m_i, \quad \forall i \in \{1, \ldots, 9\}$$

Estas tangentes determinan que cada segmento del spline, definido en los intervalos $[x_i, x_{i+1}]$, sea un polinomio cúbico que pasa por (x_i, y_i) y (x_{i+1}, y_{i+1}), con derivadas coincidentes con m_i y m_{i+1}, respectivamente.

El resultado se muestra en la Figura 4.13, donde se observa una curva suave y creciente que pasa por los puntos de penalización indicados en verde. La pendiente inicial es suave hasta alcanzar los 1748,92 km, aumentando de forma más pronunciada hasta los 5000 km, y posteriormente creciendo con agresividad hasta los 21 000 km, donde se alcanza una penalización extrema. Esta curva ilustra fielmente el objetivo de la métrica: recompensar con indulgencia las predicciones "suficientemente cercanas", y penalizar de forma progresivamente más severa conforme aumenta el error a partir de ese umbral de suficiencia.

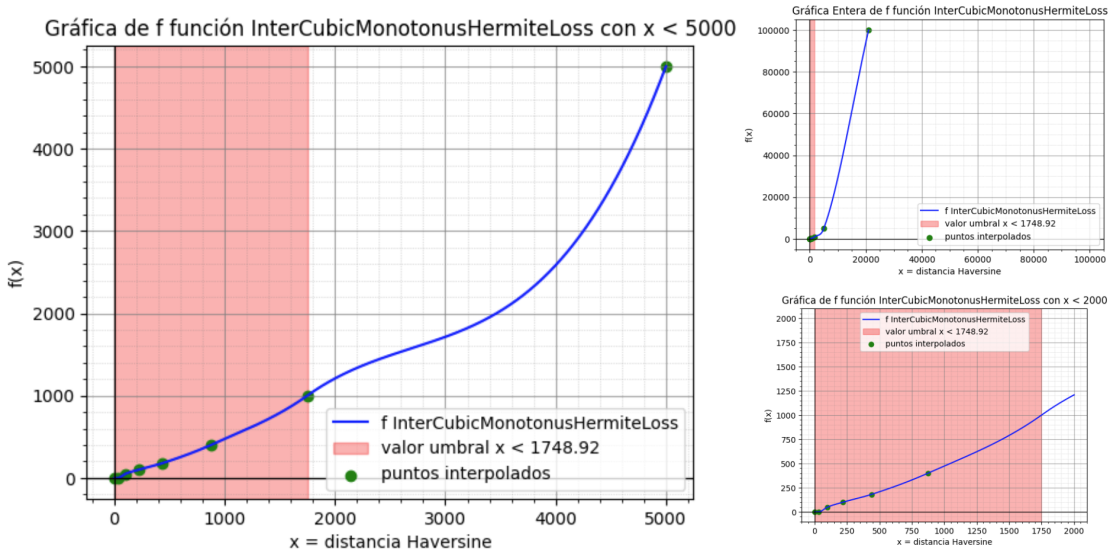

Figura 4.13: Gráfica de la Interpolación Cúbica Monótona de Hermite (InterCubicMonotonusHermiteLoss)

4.3.5.7. Enfoque de Precisión Exigente: GeoscoreModificadoLoss

Esta métrica se fundamenta en una filosofía clara: incentivar especialmente las buenas predicciones penalizando con mayor severidad los errores cuando se está muy cerca del objetivo. Cuanto más próxima esté la predicción, mayor será el peso del error cometido. Por el contrario, a medida que la distancia del error aumenta, las penalizaciones deben suavizarse progresivamente hasta estabilizarse. Esta característica busca evitar castigos excesivos cuando el error ya es considerable, permitiendo así que la red neuronal concentre sus esfuerzos en mejorar aquellas predicciones que están cerca de ser correctas.

En definitiva, esta lógica parte de la intuición de que estar "muy cerca" del resultado es mucho más valioso que simplemente estar "algo menos lejos". Para trasladar esta noción al diseño de la función de pérdida, se definen manualmente penalizaciones específicas para ciertas distancias clave, tal y como se muestra en la Tabla 4.9.

Índice	**Distancia Real (x_i)**	**Distancia métrica (y_i)**	**Descripción**
$i = 1$	0 km	0	Distancia perfecta
$i = 2$	50 km	150	Casi dos "ciudades"
$i = 3$	100 km	300	Varias "ciudades" de separación
$i = 4$	218,615 km ($\frac{1}{2} \cdot$ DMP)	655,845	Un cuarto de un país
$i = 5$	437,23 km (DMP)	1311,69	Mitad de un país
$i = 6$	874,46 km ($2 \cdot$ DMP)	2623,38	Un país completo
$i = 7$	1748,92 km ($4 \cdot$ DMP)	4372,3	Dos países
$i = 8$	5000 km	6500	Penalización fuerte
$i = 9$	$>$ 5000 km	$>$ 6500	Valores extremos

Tabla 4.9: Penalizaciones en la métrica GeoscoreModificadoLoss

La lógica detrás de estas penalizaciones consiste en comenzar con un crecimiento muy agresivo para distancias próximas a cero, representado por la función $f_1(x) = 3x$, cuya pendiente es $m_1 = 3$. Esta regla se aplica para $i \in \{1, 2, 3, 4, 5, 6\}$, asegurando así un incremento rápido en esa región inicial, que abarca desde distancias perfectas de 0 km hasta distancias comparables a la extensión de un país ($2 \cdot$ DMP). A partir de $i = 6$, se busca suavizar la pendiente mediante una nueva función $f_2(x) = 2x + n$, con pendiente $m_2 = 2$. Para garantizar la continuidad de la curva, se exige que $f_2(x)$ pase por el punto (x_6, y_6), lo que implica que $n = f_1(x_6) - 2x_6 = 874{,}46$. Por tanto, la función resultante es $f_2(x) = 2x + 874{,}46$. Esto permite calcular $y_7 = f_2(x_7) = 4372{,}3$. A continuación, se asigna de forma aproximada $y_8 = 6500$ para la distancia $x_8 = 5000$, buscando mantener la coherencia con esa tendencia de atenuar progresivamente las pendientes. Esta elección se justifica observando que la pendiente entre los puntos (x_7, y_7) y (x_8, y_8) es $m_3 = \frac{y_8 - y_7}{x_8 - x_7} = 0{,}654$, lo que confirma la progresiva reducción en la tasa de penalización.

En resumen, se observa una disminución suave entre tres tramos: uno inicial con pendiente $m_1 = 3$, luego $m_2 = 2$ y seguido por $m_3 = 0{,}654$, como se muestra en la

Figura 4.14.

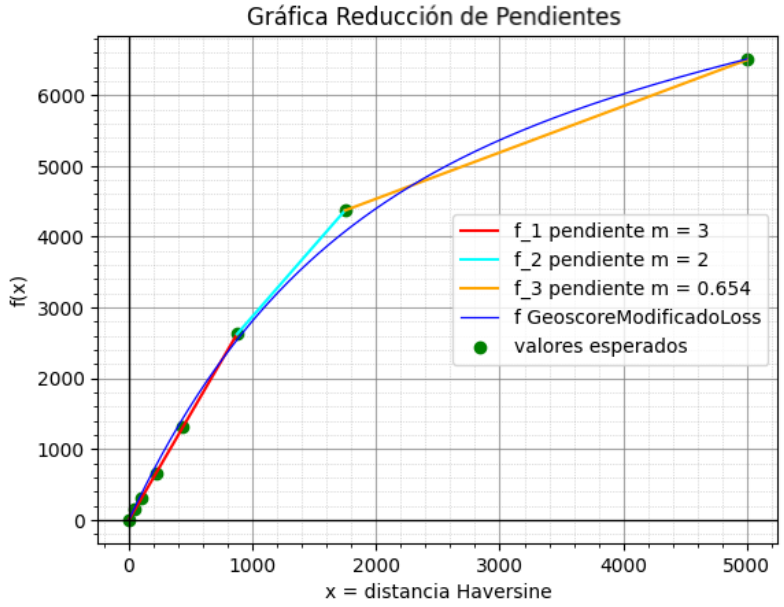

Figura 4.14: Gráfica Reducción de Pendientes

A partir de estas ideas, se considera reutilizar el Interpolador Cúbico Monótono de Hermite. No obstante, dado que este resulta complejo de aplicar directamente, y que la idea de comportamiento propuesta es similar al de la *GeoScoreLoss*, se opta por tomar esta última como base e introducirle ligeras modificaciones. De este modo, se obtiene una alternativa más sencilla, diferenciable y alineada con la filosofía buscada.

Primero, se observa que $g_1(x) = 5000 - 5000 \cdot e^{-15 \cdot \frac{x}{20037,5}}$ guarda un comportamiento muy similar a $f_1(x) = 3x$ dentro de un rango inicial de $[0, 2700] \times [0, 2700]$, donde se concentra la mayoría de los errores iniciales. Para evitar que la función se estabilice en una asíntota horizontal constante (lo que implicaría derivada nula), se introduce:

$$h_k(x) = (5000 + kx) - (5000 + kx) \cdot e^{-15 \cdot \frac{x}{20037,5}}$$

Esta nueva forma asegura que, conforme el segundo término tiende a cero, la función converja a una recta con pendiente k, evitando así la planitud indeseada.

Se analiza entonces qué valor de k resulta más adecuado. Dado que queremos seguir la lógica de suavizar las penalizaciones, no tiene sentido que $k > m_3$. Se prueban los valores $k \in \left\{\frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right\}$. Según la Figura 4.15:

- Con $k = \frac{1}{2}$ se aproxima bien (x_7, y_7) con $(x_7, 4288{,}2)$, pero queda muy lejos de (x_8, y_8) con $(x_8, 7322{,}37)$.
- Con $k = \frac{1}{3}$, aunque el punto (x_7, y_7) queda algo bajo con $(x_7, 4075{,}42)$, se acierta mucho mejor en (x_8, y_8) con $(x_8, 6508{,}78)$, logrando una reducción más realista.

- Con $k = \frac{1}{4}$ se obtienen resultados similares a $k = \frac{1}{3}$ pero con errores mayores tanto en (x_7, y_7) como en (x_8, y_8).

Por tanto, se elige $k = \frac{1}{3}$ como el valor óptimo, ya que ofrece un buen equilibrio entre fidelidad al crecimiento deseado y suavidad en la región de error elevado.

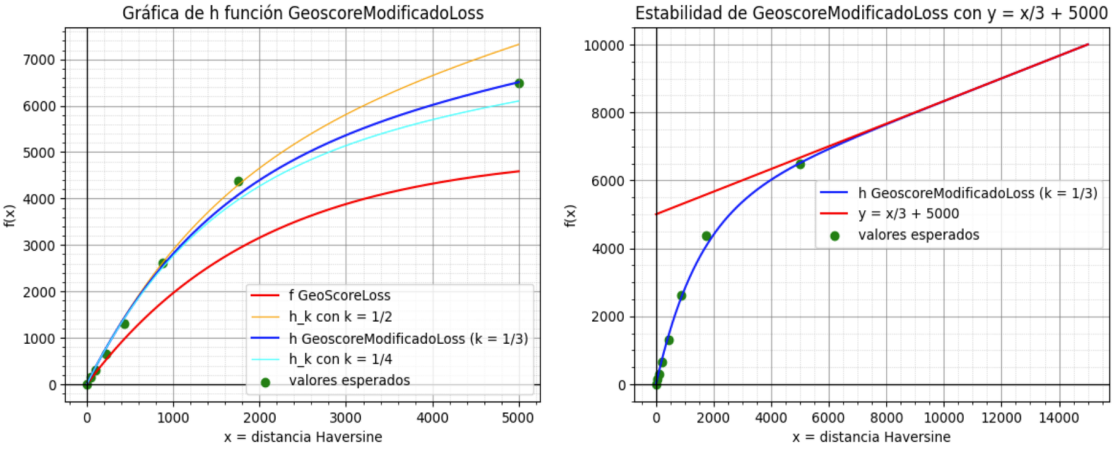

Figura 4.15: Gráfica de la función final GeoScoreModificadoLoss con $k = \frac{1}{3}$

Finalmente, con la expresión:

$$h_{1/3}(x) = \left(5000 + \frac{x}{3}\right) - \left(5000 + \frac{x}{3}\right) \cdot e^{-15 \cdot \frac{x}{20037{,}5}}$$

se logra implementar de manera diferenciable y eficiente la idea que inspira esta propuesta: penalizar fuertemente los errores cuando se está cerca del objetivo, y suavizar gradualmente las penalizaciones a medida que la distancia crece. Además, a diferencia de enfoques anteriores, evita una estabilización constante que pueda generar derivadas nulas, lo cual podría dificultar el entrenamiento. Y es que en este caso, la penalización converge suavemente hacia $y = \frac{x}{3} + 5000$, tal y como se ilustra en la Figura 4.15.

4.3.6. Optimizador

El optimizador (ver Sección 3.4.4) es esencial para determinar cómo se van a actualizar los pesos de la red neuronal durante el entrenamiento. En el caso de todos los modelos desarrollados en este trabajo, se ha empleado el optimizador *Adam*, implementado en la librería `PyTorch`, como ya se había mencionado en la Sección 4.3.4.

Adam (Kingma y Ba, 2014) es un algoritmo de optimización eficiente basado en gradientes de primer orden, diseñado específicamente para escenarios con funciones objetivo estocásticas, es decir, funciones objetivo no deterministas cuya evaluación incorpora componentes aleatorias o probabilísticas. Este es precisamente nuestro caso: aunque la función de pérdida sea determinista en su formulación, el uso de *Mini-Batch Gradient Descent* implica que su evaluación depende de subconjuntos aleatorios de los datos. Esta aleatoriedad proviene del parámetro `shuffle` activado

en el `Dataloader` de PyTorch, el cual reorganiza aleatoriamente los lotes en cada época. Como resultado, aunque cada evaluación individual de cada *mini-batch* sea determinista, el proceso global de optimización presenta un comportamiento estocástico.

El nombre *Adam* proviene de *Adaptive Moment Estimation*, es decir, de su capacidad para utilizar estimaciones de los momentos de primer y segundo orden de los gradientes para ajustar adaptativamente las tasas de aprendizaje de cada parámetro de manera individual. Esta característica resulta especialmente útil en redes profundas, donde la escala de los gradientes puede variar significativamente entre capas.

La elección de *Adam* en el trabajo se justifica por varias razones prácticas:

- Es fácil de implementar, computacionalmente eficiente y requiere poca memoria. Esto resulta crucial si ya tenemos en cuenta que debido a usar tamaños de batch de 64, es necesario optimizar los recursos para que el entrenamiento sea viable y eficiente en el hardware disponible.

- Funciona adecuadamente en problemas de gran escala, tanto en cantidad de datos como en número de parámetros. Esto es especialmente relevante en nuestro escenario, donde se entrena sobre un conjunto de datos masivo (más de 4 millones de imágenes) y se utiliza una `MLP` densamente conectada, lo que genera una gran cantidad de pesos a optimizar.

- Es compatible con gradientes dispersos (*sparse gradients*). Los gradientes dispersos se producen cuando solo un pequeño número de parámetros del modelo tienen gradientes diferentes de cero, mientras que la mayoría de ellos tienen gradientes cercanos a cero. Una situación común en redes profundas con activaciones ReLU, ya que estas generan gradientes nulos para entradas negativas. *Adam* maneja bien este tipo de gradientes gracias a un término de corrección del sesgo, lo que evita actualizaciones excesivamente grandes de los pesos que podrían hacer divergir el entrenamiento.

- Se ha demostrado empíricamente que *Adam* es robusto frente a una amplia variedad de problemas de optimización no convexos en el campo del aprendizaje automático. El problema de geolocalización de una imagen se está resolviendo usando una red neuronal profunda donde la función a optimizar tiende a ser no convexa debido a la interacción no lineal entre las capas introducida por las funciones de activación no lineales.

Las tasas de aprendizaje que *Adam* genera están acotadas por un hiperparámetro base, ya mencionado en la Sección 4.3.4. Además, el algoritmo implementa de forma natural una disminución progresiva del tamaño del paso (*step size annealing*), ya que las actualizaciones se basan en promedios móviles de los momentos del gradiente, lo cual reduce progresivamente su magnitud a medida que el entrenamiento avanza. Esta dinámica permite que, al inicio del entrenamiento, el modelo realice pasos relativamente grandes, gracias a la mayor magnitud de los gradientes iniciales, lo que facilita un avance rápido hacia regiones prometedoras del espacio de soluciones.

A medida que el modelo mejora y los gradientes se estabilizan, las actualizaciones se vuelven más pequeñas, lo que permite afinar la solución y evitar que el modelo sobrepase un mínimo local o global.

4.3.7. Listado de Modelos a Entrenar

Llegados a este punto y tras revisar todas las alternativas de configuración que vamos a considerar, pasamos a detallar los 36 modelos diferentes que se van a entrenar e indicar todas sus características, primero especificando los parámetros comunes, véase Tabla 4.10, y luego los distintivos, véase Tabla 4.11.

Parámetro	Valor
Arquitectura	CLIP-ViT[5] $\longrightarrow$ MLP[6]
Arquitectura Base MLP	$512 \rightarrow 256 \xrightarrow{\text{ReLU}} 128 \xrightarrow{\text{ReLU}} 2$
Paciencia	5
Tasa de Aprendizaje Base	0,001
Batch Size	64
Optimizador	Adam (lr=0,001)

Tabla 4.10: Parámetros comunes en todos los modelos

4.4. Entrenamiento de los Modelos

Una vez definidos los submodelos y los parámetros correspondientes, se procede al entrenamiento de cada uno de ellos. El modelo `CLIP-ViT` ya ha sido previamente entrenado por `OpeanAI`, por lo que no se realizará un reentrenamiento del mismo, sino que se utilizará tal cual. En consecuencia, el enfoque se centrará exclusivamente en el entrenamiento de los modelos `MLP`, empleando para ello los conjuntos de entrenamiento y validación.

Para este propósito, todas las imágenes contenidas en el conjunto `LMDB`, tanto de entrenamiento como de validación, fueron procesadas mediante el modelo `CLIP-ViT` tras aplicar el preprocesamiento necesario. En concreto, las imágenes se redimensionaron a 224×224 píxeles, dado que dicho tamaño es el requerido por los modelos `ViT-B/32` y `ViT-B/16`. Estos modelos generaron *embeddings*, vectores de características, de dimensión $d = 512$, los cuales, tras una normalización L_2, han sido utilizados como entrada para alimentar los respectivos modelos de geolocalización durante su entrenamiento. Los tiempos requeridos para la generación de dichos *embeddings* se muestran en la Tabla 4.5, sumando en total más de 146 horas.

[5]Image Encoder especificado en Sección 4.3.2
[6]Módulo de Geolocalización especificado en Sección 4.3.3

	Módelo	Img Encoder	Métrica	Func. última Capa
1	GeoLocB32-MSEc	CLIP-ViT-B/32	MSELoss (MSE)	clamp
2	GeoLocB32-MSEt	CLIP-ViT-B/32	MSELoss (MSE)	tanh
3	GeoLocB32-MSEl	CLIP-ViT-B/32	MSELoss (MSE)	logística
4	GeoLocB32-HSc	CLIP-ViT-B/32	HaversineLoss (HS)	clamp
5	GeoLocB32-HSt	CLIP-ViT-B/32	HaversineLoss (HS)	tanh
6	GeoLocB32-HSl	CLIP-ViT-B/32	HaversineLoss (HS)	logística
7	GeoLocB32-GSc	CLIP-ViT-B/32	GeoscoreLoss (GS)	clamp
8	GeoLocB32-GSt	CLIP-ViT-B/32	GeoscoreLoss (GS)	tanh
9	GeoLocB32-GSl	CLIP-ViT-B/32	GeoscoreLoss (GS)	logística
10	GeoLocB32-LSc	CLIP-ViT-B/32	LinealScoreLoss (LS)	clamp
11	GeoLocB32-LSt	CLIP-ViT-B/32	LinealScoreLoss (LS)	tanh
12	GeoLocB32-LSl	CLIP-ViT-B/32	LinealScoreLoss (LS)	logística
13	GeoLocB32-ICMHc	CLIP-ViT-B/32	InterCubicMonotonusHermiteLoss (ICMH)	clamp
14	GeoLocB32-ICMHt	CLIP-ViT-B/32	InterCubicMonotonusHermiteLoss (ICMH)	tanh
15	GeoLocB32-ICMHl	CLIP-ViT-B/32	InterCubicMonotonusHermiteLoss (ICMH)	logística
16	GeoLocB32-GSMc	CLIP-ViT-B/32	GeoscoreModificadoLoss (GSM)	clamp
17	GeoLocB32-GSMt	CLIP-ViT-B/32	GeoscoreModificadoLoss (GSM)	tanh
18	GeoLocB32-GSMl	CLIP-ViT-B/32	GeoscoreModificadoLoss (GSM)	logística
19	GeoLocB16-MSEc	CLIP-ViT-B/16	MSELoss (MSE)	clamp
20	GeoLocB16-MSEt	CLIP-ViT-B/16	MSELoss (MSE)	tanh
21	GeoLocB16-MSEl	CLIP-ViT-B/16	MSELoss (MSE)	logística
22	GeoLocB16-HSc	CLIP-ViT-B/16	HaversineLoss (HS)	clamp
23	GeoLocB16-HSt	CLIP-ViT-B/16	HaversineLoss (HS)	tanh
24	GeoLocB16-HSl	CLIP-ViT-B/16	HaversineLoss (HS)	logística
25	GeoLocB16-GSc	CLIP-ViT-B/16	GeoscoreLoss (GS)	clamp
26	GeoLocB16-GSt	CLIP-ViT-B/16	GeoscoreLoss (GS)	tanh
27	GeoLocB16-GSl	CLIP-ViT-B/16	GeoscoreLoss (GS)	logística
28	GeoLocB16-LSc	CLIP-ViT-B/16	LinealScoreLoss (LS)	clamp
29	GeoLocB16-LSt	CLIP-ViT-B/16	LinealScoreLoss (LS)	tanh
30	GeoLocB16-LSl	CLIP-ViT-B/16	LinealScoreLoss (LS)	logística
31	GeoLocB16-ICMHc	CLIP-ViT-B/16	InterCubicMonotonusHermiteLoss (ICMH)	clamp
32	GeoLocB16-ICMHt	CLIP-ViT-B/16	InterCubicMonotonusHermiteLoss (ICMH)	tanh
33	GeoLocB16-ICMHl	CLIP-ViT-B/16	InterCubicMonotonusHermiteLoss (ICMH)	logística
34	GeoLocB16-GSMc	CLIP-ViT-B/16	GeoscoreModificadoLoss (GSM)	clamp
35	GeoLocB16-GSMt	CLIP-ViT-B/16	GeoscoreModificadoLoss (GSM)	tanh
36	GeoLocB16-GSMl	CLIP-ViT-B/16	GeoscoreModificadoLoss (GSM)	logística

Tabla 4.11: Parámetros diferentes entre los modelos

Una vez completado el entrenamiento de todos los modelos `MLP` (véase Tabla 4.12 al final de esta sección), se procede a analizar la evolución del error incurrido tanto en el conjunto de entrenamiento como en el de validación.

Las gráficas presentan en color azul la media del error en cada época del conjunto de entrenamiento y en color rojo la media del error del conjunto de validación. Se muestra un ejemplo de gráfica en la Figura 4.16 que se corresponde a la evolución del error durante el entrenamiento del modelo `GeoLocB16-GSMl`. Las gráficas correspondientes al resto de los modelos, tanto para los de `CLIP-ViT-B/32` como `CLIP-ViT-B/16`, se incluyen en el Apéndice A.

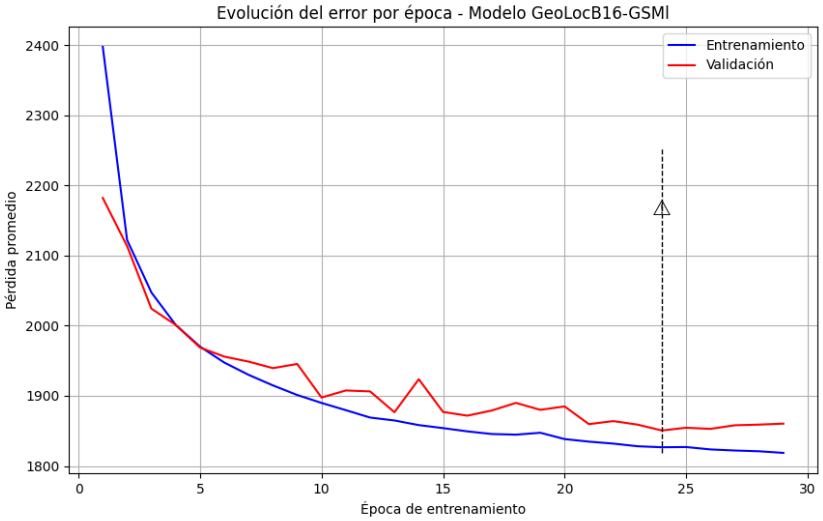

Figura 4.16: Ejemplo Gráfica Evolución del Error durante el Entrenamiento en el modelo `GeoLocB16-GSMl`

Se observa que, en la mayoría de los casos, ambas curvas descienden de forma paralela y mantienen una proximidad considerable durante prácticamente todo el entrenamiento. Esto resulta en una diferencia relativa reducida entre el error de validación y el de entrenamiento en las fases finales, es decir, que no se han separado demasiado, lo que constituye un indicador favorable de que el modelo no ha sufrido sobreajuste significativo. Del análisis de estos comportamientos, podemos identificar dos patrones predominantes (siendo el primero considerablemente más frecuente):

- Casos donde el punto de detención del entrenamiento coincide con el momento en que el error de validación comienza, durante varias épocas consecutivas (superando en ocasiones las 5 épocas), a estancarse o incluso aumentar, mientras que el error de entrenamiento continúa disminuyendo, generando una divergencia que cada vez se hace mayor entre ambas pérdidas. Esta separación se manifiesta con distinta intensidad según el modelo, pero representa el punto idóneo (*sweet spot*) que se buscaba con el *Early Stopping* para evitar el sobreajuste. Ejemplos notables de este comportamiento se observan en los modelos `GeoLocB32-HSt`, `GeoLocB32-ICMHc`, `GeoLocB16-HSc`, `GeoLocB16-LSl` y `GeoLocB16-GSMl`, entre otros. Se pueden ver las gráficas en cuestión en el apéndice en las Figuras A.2, A.5, A.8, A.10 y A.12 respectivamente.

- Casos donde ambas curvas de error mantienen una similitud extremadamente alta y experimentan: (i) un estancamiento prolongado durante numerosas épocas, o (ii) una situación donde, previamente a la detención, el error de validación muestra oscilaciones irregulares con mejoras ocasionales pero sin una tendencia clara de optimización. El primer fenómeno puede interpretarse como un indicio de que el modelo ha alcanzado su límite de capacidad de aprendizaje o ha alcanzado la convergencia hacia un mínimo local del que no logra escapar. En el segundo escenario, es fundamental actuar con precaución para evitar el fenómeno descrito en la Sección 3.4.1, donde la métrica de validación puede experimentar mejoras puntuales que no reflejan necesariamente una mayor capacidad de generalización. Dicho de otro modo, existe el riesgo de que el modelo se sobreajuste también al conjunto de validación, de manera análoga al sobreajuste que ocurre con los datos de entrenamiento cuando el proceso se prolonga excesivamente. Este fenómeno puede manifestarse cuando una configuración específica, ya excesivamente adaptada a los datos de entrenamiento, coincide en ajustarse inusualmente bien al conjunto de validación, produciendo una reducción engañosa de la pérdida. Ejemplos representativos del caso de similitud extremadamente alta mencionado incluyen `GeoLocB32-GSc`, `GeoLocB16-GSc` (correspondientes al escenario (i) mencionado) y `GeoLocB32-GSl` (correspondiente al escenario (ii)). Se pueden ver las gráficas en cuestión en el apéndice en las Figuras A.3 y A.9 respectivamente.

Por lo expuesto anteriormente, se evidencia que la implementación de una paciencia de 5 en el mecanismo de *Early Stopping* ha resultado efectiva. Esta configuración ha sido determinante para prevenir divergencias significativas entre ambas métricas de error (especialmente en los casos del primer tipo), evitando así potenciales situaciones de sobreajuste. Asimismo, ha proporcionado una respuesta adecuada en aquellos escenarios donde las curvas de error mostraban un comportamiento paralelo y próximo entre sí, con cierto estancamiento. En estos casos, el error de validación experimentaba fluctuaciones antes de la detención, sin una tendencia clara de optimización, o bien el estancamiento prolongado indicaba que se había alcanzado un buen rendimiento y el estancamiento era simplemente el techo de aprendizaje del modelo. Como se especificó en la Sección 4.3.4, este valor de paciencia se estableció tras observar que el error en validación típicamente disminuía con rapidez durante las primeras épocas para posteriormente estabilizarse, comportamiento que, como se ha constatado, se produce en la gran mayoría de los casos analizados.

También encontramos un caso excepcional con el modelo `GeoLocB32-ICMHt` (ver en el apéndice Figura A.5), que presenta una alta variabilidad y fluctuaciones notables, especialmente en la curva de validación, aunque también se observan ciertas oscilaciones en el entrenamiento. Este comportamiento podría deberse a alguna anomalía ocurrida durante el proceso de entrenamiento, como una distribución inusual y desafortunada en los datos (como que justamente juntase aleatoriamente imágenes muy mal localizadas a la vez), problemas de randomización o alguna irregularidad en los lotes de entrada. A pesar de estas fluctuaciones, los errores finales de entrenamiento y validación resultan bastante similares, lo cual sugiere que, aunque los resultados puedan parecer anómalos, no hay indicios claros de sobreajuste.

	Módelo	T. Image Encoder[7]	T. MLP[8]	Épocas MLP	T. Total Exacto[9]	T. Total Approx
1	GeoLocB32-MSEc	29h 51min 44,9s	3h 15min 16,39s	37	33h 7min 1,29s	≈ 33h
2	GeoLocB32-MSEt	29h 51min 44,9s	1h 44min 31,92s	22	31h 36min 16,82s	≈ 32h
3	GeoLocB32-MSEl	29h 51min 44,9s	1h 58min 57,57s	23	31h 50min 42,47s	≈ 32h
4	GeoLocB32-HSc	29h 51min 44,9s	10h 38min 13,54s	75	40h 29min 58,44s	≈ 40h
5	GeoLocB32-HSt	29h 51min 44,9s	6h 5min 56,13s	42	35h 57min 41,03s	≈ 36h
6	GeoLocB32-HSl	29h 51min 44,9s	3h 36min 12,72s	25	33h 27min 57,62s	≈ 33h
7	GeoLocB32-GSc	29h 51min 44,9s	8h 35min 39,56s	38	38h 27min 24,46s	≈ 38h
8	GeoLocB32-GSt	29h 51min 44,9s	3h 58min 19,51s	33	33h 50min 4,41s	≈ 34h
9	GeoLocB32-GSl	29h 51min 44,9s	3h 54min 4,52s	24	33h 45min 49,42s	≈ 34h
10	GeoLocB32-LSc	29h 51min 44,9s	19h 31min 20,54s	106	49h 23min 5,44s	≈ 49h
11	GeoLocB32-LSt	29h 51min 44,9s	7h 24min 42,75s	30	37h 16min 27,65s	≈ 37h
12	GeoLocB32-LSl	29h 51min 44,9s	9h 12min 23,77s	45	39h 4min 8,67s	≈ 39h
13	GeoLocB32-ICMHc	29h 51min 44,9s	29h 43min 8,96s	50	59h 34min 53,86s	≈ 60h
14	GeoLocB32-ICMHt	29h 51min 44,9s	12h 11min 20,00s	19	42h 3min 4,90s	≈ 42h
15	GeoLocB32-ICMHl	29h 51min 44,9s	15h 33min 43,57s	30	45h 25min 28,47s	≈ 45h
16	GeoLocB32-GSMc	29h 51min 44,9s	19h 37min 23,01s	95	49h 29min 7,91s	≈ 49h
17	GeoLocB32-GSMt	29h 51min 44,9s	6h 0min 56,10s	24	35h 52min 41,00s	≈ 36h
18	GeoLocB32-GSMl	29h 51min 44,9s	9h 40min 35,05s	45	39h 32min 19,95s	≈ 40h
19	GeoLocB16-MSEc	116h 27min 46,4s	3h 46min 34,20s	37	120h 14min 20,6s	≈ 120h
20	GeoLocB16-MSEt	116h 27min 46,4s	1h 32min 14,31s	14	118h 0min 0,71s	≈ 118h
21	GeoLocB16-MSEl	116h 27min 46,4s	2h 26min 28,84s	29	118h 54min 15,24s	≈ 119h
22	GeoLocB16-HSc	116h 27min 46,4s	22h 37min 6,81s	71	139h 4min 53,21s	≈ 139h
23	GeoLocB16-HSt	116h 27min 46,4s	4h 11min 16,20s	29	120h 39min 2,60s	≈ 121h
24	GeoLocB16-HSl	116h 27min 46,4s	4h 24min 0,99s	29	120h 51min 47,39s	≈ 121h
25	GeoLocB16-GSc	116h 27min 46,4s	15h 40min 53,97s	55	132h 8min 40,37s	≈ 132h
26	GeoLocB16-GSt	116h 27min 46,4s	2h 28min 37,83s	16	118h 56min 24,23s	≈ 119h
27	GeoLocB16-GSl	116h 27min 46,4s	4h 11min 42,02s	27	120h 39min 28,42s	≈ 121h
28	GeoLocB16-LSc	116h 27min 46,4s	18h 49min 13,99s	71	135h 17min 0,39s	≈ 135h
29	GeoLocB16-LSt	116h 27min 46,4s	3h 3min 27,18s	21	119h 31min 13,58s	≈ 120h
30	GeoLocB16-LSl	116h 27min 46,4s	7h 57min 48,63s	28	124h 25min 35,03s	≈ 124h
31	GeoLocB16-ICMHc	116h 27min 46,4s	24h 45min 0,15s	41	141h 12min 46,55s	≈ 141h
32	GeoLocB16-ICMHt	116h 27min 46,4s	20h 34min 10,58s	25	137h 1min 56,98s	≈ 137h
33	GeoLocB16-ICMHl	116h 27min 46,4s	11h 38min 10,22s	13	128h 5min 56,62s	≈ 128h
34	GeoLocB16-GSMc	116h 27min 46,4s	10h 16min 22,49s	62	126h 44min 8,89s	≈ 127h
35	GeoLocB16-GSMt	116h 27min 46,4s	4h 24min 25,88s	25	120h 52min 12,28s	≈ 121h
36	GeoLocB16-GSMl	116h 27min 46,4s	4h 10min 18,54s	24	120h 38min 4,94s	≈ 121h

Tabla 4.12: Duración Entrenamiento de los Modelos

[7]**T. Image Encoder:** tiempo que ha tardado el *Image Encoder* en sacar todos los *embeddings* para entrenamiento (recordar que los *embeddings* de cada tipo de codificador usado solo se sacaron una vez)

[8]**T. MLP:** tiempo que ha tardado el Modelo de Geolocalización en entrenarse

[9]**T. Total Exacto:** tiempo total exacto que se ha tardado para generar el modelo y que este operativo (será la suma de **T. Image Encoder** y **T. MLP**)

4.5. Evaluación del Rendimiento y Selección del Mejor Modelo

A continuación, se procederá a la evaluación de los distintos modelos desarrollados, utilizando para ello el conjunto de test disponible. Se analizarán los resultados obtenidos y se comparará el rendimiento de cada modelo.

Aunque los modelos se hayan entrenado con diferentes funciones de pérdida, todos serán evaluados utilizando la distancia de Haversine como métrica principal. Al final, esta métrica resulta especialmente adecuada para la tarea planteada en este trabajo, puesto que permite medir de forma precisa la distancia entre dos ubicaciones geográficas (la real y la predicha), siendo, por tanto, la más representativa para valorar la calidad de las predicciones realizadas a partir de imágenes.

Podría considerarse que entrenar los modelos con distintas funciones de pérdida carece de utilidad si posteriormente todos serán evaluados bajo la misma métrica. Sin embargo, esta diversidad de funciones de error introduce diferentes estrategias de aprendizaje en la red neuronal que dan más importancia a unas penalizaciones u a otras, afectando de manera indirecta al proceso de optimización de los pesos. Algunas funciones de pérdida penalizan con mayor severidad los errores atípicos, mientras que otras fomentan la obtención de predicciones muy cercanas a la ubicación real frente a predicciones menos lejanas, priorizando la precisión extrema sobre una reducción general del error medio. En consecuencia, se exploran distintas formas de interpretar y aprender las características geográficas codificadas en los vectores de representación de las imágenes, lo que puede resultar en que alguna de esas estrategias obtenga un mejor reconocimiento de patrones relevantes para la tarea de regresión de coordenadas.

Aunque la distancia de Haversine será la métrica utilizada para la evaluación, el criterio para seleccionar el mejor modelo no se basará, como suele ser usual, en minimizar la media del error cometido. Además de buscar la mayor precisión posible, se pretende que el modelo sea capaz de identificar patrones geográficos generales de manera robusta.

Se busca un equilibrio entre dos objetivos: obtener buenas predicciones de manera generalizada y asegurar que el modelo consiga acertar muchas veces y de manera muy precisa, incluso si ocasionalmente comete errores graves. Esto último implica aceptar que algunos errores aislados puedan ser elevados si, a cambio, se logra aumentar la probabilidad de obtener predicciones correctas. Por ello, no se elegirá el modelo en función de la media del error en el conjunto de test.

Para ilustrar el porqué, consideremos dos modelos con cuatro predicciones cada uno. El primer modelo obtiene tres predicciones exactas (0 km de error) y una predicción errónea a 20 000 km, mientras que el segundo modelo tiene una predicción exacta y tres predicciones con errores de 6000 km cada una. Calculando las medias:

$$\text{Media Modelo 1} = \frac{0 + 0 + 0 + 20000}{4} = 5000 \text{ km}$$

$$\text{Media Modelo 2} = \frac{0 + 6000 + 6000 + 6000}{4} = 4500 \text{ km}$$

Aunque el segundo modelo presenta una media de error inferior, el primero logra un mayor número de predicciones exactas. Por tanto, desde el enfoque de este trabajo, el primer modelo sería preferible, ya que falla menos veces, considerando como fallo aquellas predicciones lejanas, sin importar tanto la magnitud del error.

Esta justificación refuerza la importancia de haber entrenado con diferentes funciones de pérdida, ya que permite crear modelos que, además de ofrecer buenas medias (y por tanto buenas predicciones en general), presenten una mayor capacidad para realizar muchas predicciones altamente precisas y minimizar el número de fallos severos.

Para poder analizar el rendimiento equilibrado de los modelos, se utilizarán dos tipos de gráficas:

- **Histogramas de errores geográficos:** se trata de gráficas que muestran la cantidad de predicciones que se encuentran a determinadas distancias (en kilómetros) de la ubicación real. Las predicciones se agrupan en intervalos de tamaño fijo, y para cada intervalo se indica cuántas imágenes del conjunto de test presentan un error dentro de ese rango. El eje de abscisas representa el error en kilómetros, mientras que el eje de ordenadas indica el número de imágenes correspondientes a cada intervalo. Esta representación permite visualizar la dispersión de los errores, proporcionando información sobre la cantidad de predicciones cercanas y lejanas respecto a las ubicaciones reales.

- **Función de Distribución Acumulada (CDF) del error geográfico:** también conocida por sus siglas en inglés *Cumulative Distribution Function*. Esta gráfica muestra la proporción acumulada de imágenes cuyo error es menor o igual a una cierta distancia. El eje de abscisas representa el error geográfico en kilómetros, y el eje de ordenadas indica la proporción acumulada de imágenes en porcentaje. Esta representación permite visualizar de manera intuitiva qué tan rápido se obtienen predicciones cercanas. Es especialmente útil para comparar modelos y seleccionar el mejor, ya que se buscará aquel cuya CDF tenga un crecimiento inicial rápido y pronunciado, y que se mantenga por encima del resto de modelos, idealmente en toda la gráfica. Un crecimiento rápido en los primeros tramos implica que el modelo acierta un mayor número de predicciones cercanas (lo que se traduce en una pendiente mayor en la gráfica inicial). Esta característica es fundamental para cumplir con el objetivo de equilibrio propuesto: lograr buenas predicciones generales (manteniéndose consistentemente por encima de los demás modelos) y, a la vez, maximizar el número de predicciones altamente precisas (reflejadas en una subida inicial más pronunciada en la CDF).

El eje de abscisas correspondiente al error geográfico en ambas representaciones gráficas abarcará desde los 0 km, que indicarían predicciones exactas, hasta los 20 100 km. Cabe destacar que esta última distancia no puede alcanzarse en la práctica, ya que, como se justificó en la Sección 4.3.5.3, la distancia máxima posible entre dos puntos en la superficie terrestre es de aproximadamente 20 037,5 km.

4.5.1. Histogramas de Errores Geográficos

Se ha considerado un tamaño de intervalo de 100 km para la representación de los histogramas. La elección de este valor es especialmente relevante para evaluar de manera precisa el rendimiento de los modelos. Divisiones demasiado amplias podrían enmascarar diferencias significativas entre predicciones buenas y malas. Por ejemplo, un intervalo amplio podría aparentar que un modelo tiene mejores resultados simplemente porque acumula más predicciones en un rango, aunque estas se encuentren más alejadas (por ejemplo, alrededor de 120 km) en comparación con otro modelo cuyas predicciones, aunque menos numerosas en ese intervalo, se situarían más cerca (alrededor de 70 km).

Esta consideración está directamente relacionada con la idea de representar adecuadamente el nivel de acierto en las predicciones. Idealmente, se utilizarían intervalos lo más pequeños posible, pero dado que el límite superior de la distancia Haversine es de aproximadamente 20100 km, intervalos significativamente menores a 100 km harían ilegible la interpretación de los datos. Por ello, se ha optado por un equilibrio que mantiene la legibilidad sin perder precisión.

Se puede ver un ejemplo de gráfica en la Figura 4.17 que se corresponde al histograma de errores geográficos cometidos en el conjunto de test con el modelo `GeoLocB16-LSl`. Los histogramas correspondientes al resto de modelos, tanto para el *Image Encoder* `CLIP-ViT-B/32` como para el `CLIP-ViT-B/16`, se incluyen en el Apéndice B.

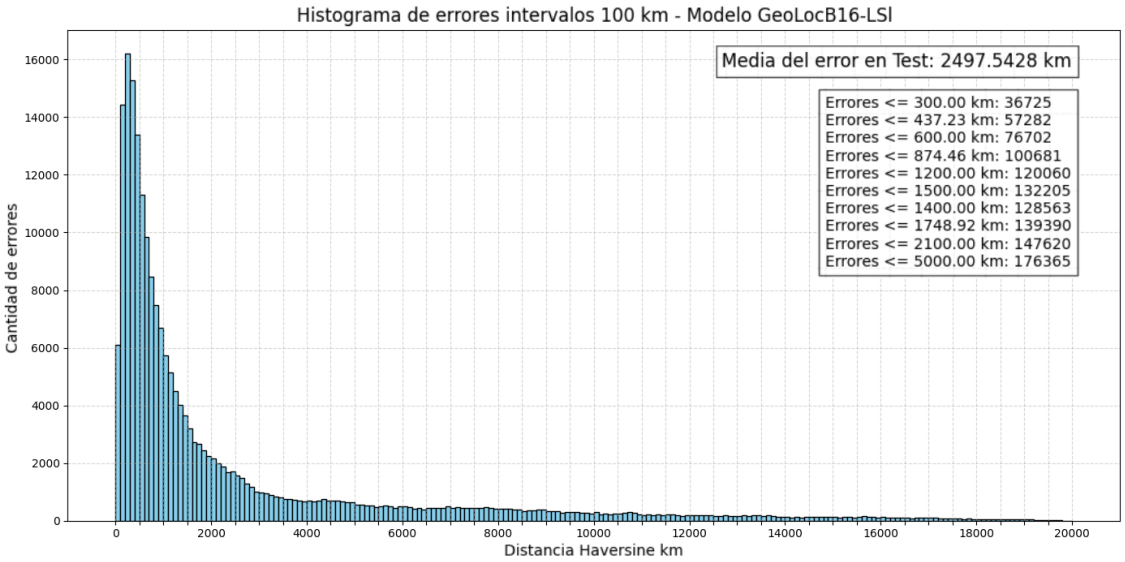

Figura 4.17: Ejemplo histograma de errores geográficos en el modelo `GeoLocB16-LSl`

En todos los histogramas se observa una notable concentración de imágenes hacia la izquierda de la gráfica, con un pico de frecuencia en las distancias Haversine más pequeñas que decrece progresivamente a medida que aumenta la distancia. Esta tendencia confirma que todos los modelos, en mayor o menor medida, han aprendido a realizar buenas predicciones sobre datos no vistos y su generalización sobre imágenes nuevas no se ha comprometido con un excesivo sobreajuste. La distribución no es aleatoria, sino que refleja un aprendizaje significativo de los patrones geográficos.

Además, en cada histograma se ha incorporado un cuadro de texto indicando el error medio de cada modelo (aunque, como se ha mencionado previamente, no será el criterio principal de evaluación, sino con fines informativos), junto con otro cuadro que resume de manera más clara y visual el número de imágenes clasificadas con errores inferiores o iguales a m km.

Los valores de error se representan en intervalos de 300 km, salvo tres valores específicos: 437,23 km, 874,46 km y 1748,92 km. Estos valores tienen un significado particular relacionado con el concepto de `DMP` introducido en la Sección 4.3.5.4, que asocia estas distancias a medidas intuitivas como la mitad de un país, un país completo y dos países, respectivamente. Esta representación adicional facilita una interpretación más intuitiva del rendimiento de los modelos en términos de localización geográfica.

4.5.2. Función de Distribución Acumulada (CDF)

Tal y como se ha explicado previamente, esta gráfica refleja de manera muy efectiva el equilibrio buscado en los modelos: predecir de forma general lo más correctamente posible, a la vez que se logra un alto número de aciertos exactos, tolerando algunas anomalías o predicciones fallidas puntuales.

En la Figura 4.18 se presentan las CDF de todos los modelos evaluados. Las subfiguras superiores corresponden a los 18 modelos basados en el *Image Encoder* `CLIP-ViT-B/32`, mientras que las inferiores representan los 18 modelos basados en el `CLIP-ViT-B/16`.

A partir de esta figura, se identifican los modelos cuya CDF se mantiene más consistentemente por encima de las demás dentro de cada grupo de `ViT-B/32` y `ViT-B/16`. Estos modelos, considerados como los candidatos más prometedores, se recogen en la Figura 4.19 para los modelos `ViT-B/32`, y en la Figura 4.20 para los modelos `ViT-B/16`. En ambas figuras, la gráfica de la izquierda muestra la CDF en todo el rango de distancias, mientras que la gráfica de la derecha presenta un *zoom* en las distancias iniciales.

Con apoyo en las gráficas de *zoom* en las distancias iniciales, se va a elegir el mejor modelo de cada grupo. El criterio seguido es seleccionar aquel modelo que, desde el inicio y durante un tramo significativo, presente la CDF más elevada, es decir, el que consigue clasificar correctamente el mayor número de imágenes a corta distancia.

Siguiendo este criterio, se identifican como mejores modelos a `GeoLocB32-LSc` para los basados en `ViT-B/32` y a `GeoLocB16-LSl` para los basados en `ViT-B/16`.

Finalmente, nos quedamos con `GeoLocB16-LSl` como el mejor modelo de todos dentro de nuestro objetivo equilibrado, ya que se puede ver en la Figura 4.21 una clara superioridad de los modelos basados en `CLIP-ViT-B/16` (en azul) frente a los que usan el `CLIP-ViT-B/32` (en rojo). Esto confirma la hipótesis inicial planteada al introducir ambos *Image Encoders*: el modelo `ViT-B/16`, al dividir las imágenes en parches más pequeños, captura mejor las características visuales de la imagen en los vectores de *embeddings* suministrados al modelo de geolocalización. Esto enriquece el espacio de entrada, facilitando la extracción de relaciones entre las características visuales y su localización geográfica.

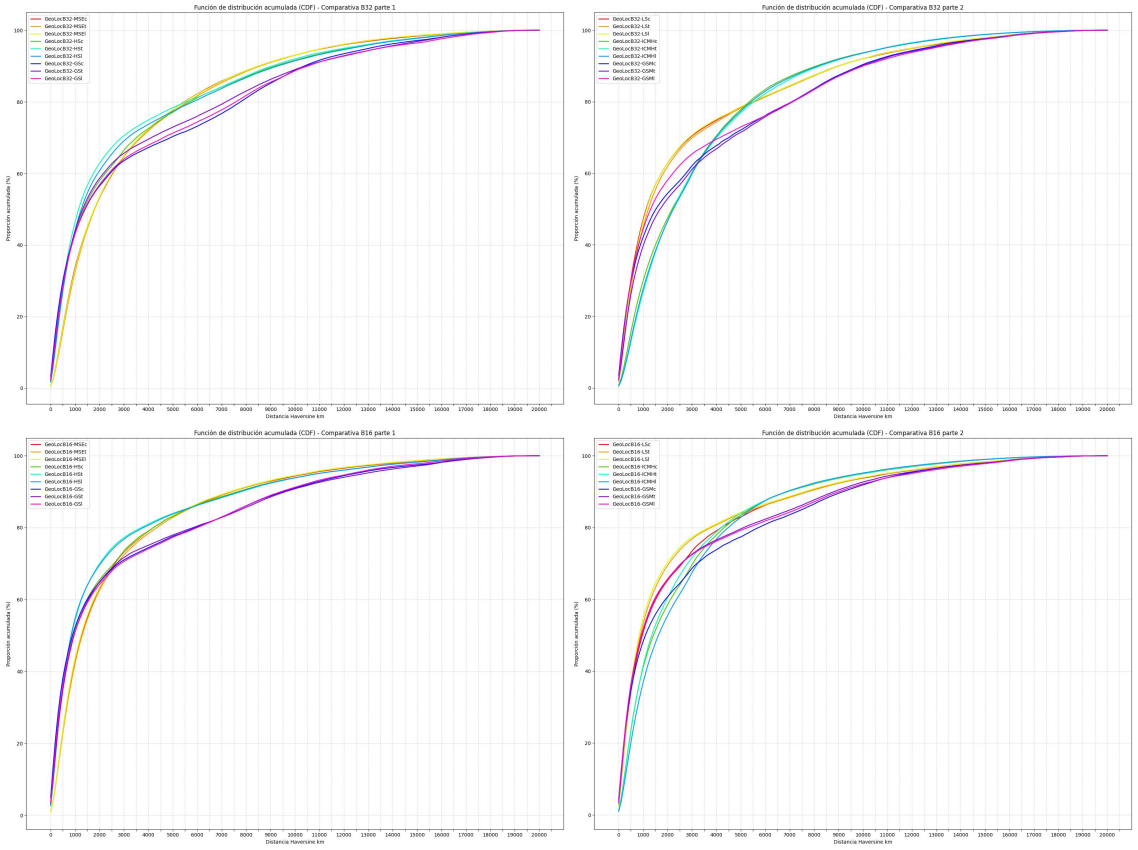

Figura 4.18: Funciones de distribución acumulada (CDF) de todos los modelos (ver imágenes ampliadas en el Apéndice B, Figuras B.13 y B.14 respectivamente)

El modelo `GeoLocB16-LS1` (cuyo histograma se puede ver en la Figura 4.17) representa el equilibrio ideal entre ambos objetivos planteados. Su elección se ha fundamentado, en primer lugar, en la identificación de los modelos cuya CDF se mantenía más alta de manera consistente (Figura 4.18), lo que garantizaba un rendimiento global sólido. En segundo lugar, dentro de estos modelos, se ha priorizado aquel que lograba mejores resultados en las distancias más cortas, reflejando así una mayor capacidad para realizar predicciones muy precisas. Este criterio responde a la idea de favorecer modelos que acierten con alta exactitud en la mayoría de los casos, incluso si ocasionalmente se producen anomalías con predicciones muy alejadas.

4.5.3. Resultados Adicionales

Los modelos entrenados con la métrica `HaversineLoss` y `LinealScoreLoss` son los que han ido apareciendo en esa búsqueda del mejor equilibrio entre buenas predicciones en general y un aseguramiento de acertar muchas veces de manera muy precisa, incluso si ocasionalmente se cometían errores muy grandes.

Sin embargo, hay otras funciones de pérdida cuyos modelos no han podido entrar en esa lucha de equilibrio para elegir el mejor, pero han presentado resultados destacables y dignos de mención gracias a esas estrategias de aprendizaje configuradas dentro de las métricas que daban más importancia a unas penalizaciones u otras.

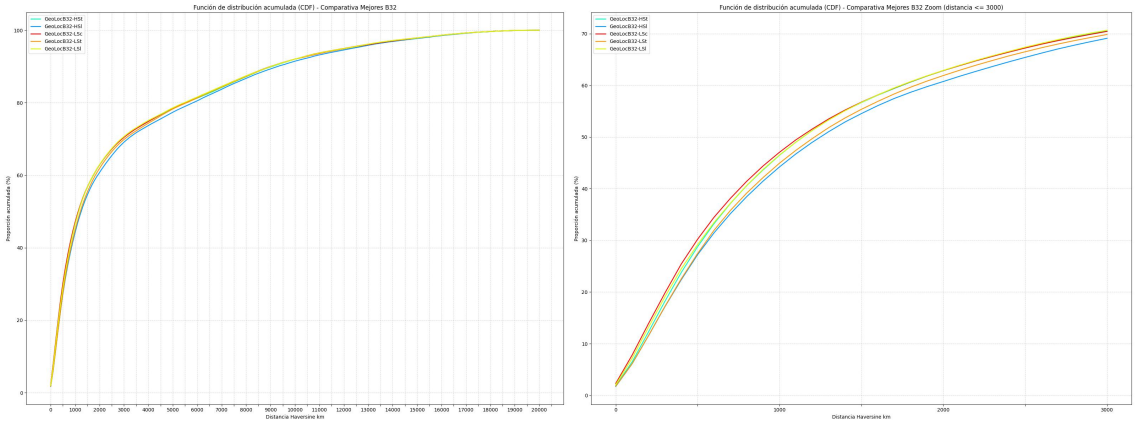

Figura 4.19: Mejores funciones de distribución acumulada de los modelos basados en `ViT-B/32` (ver imagen ampliada en el Apéndice B, Figura B.15)

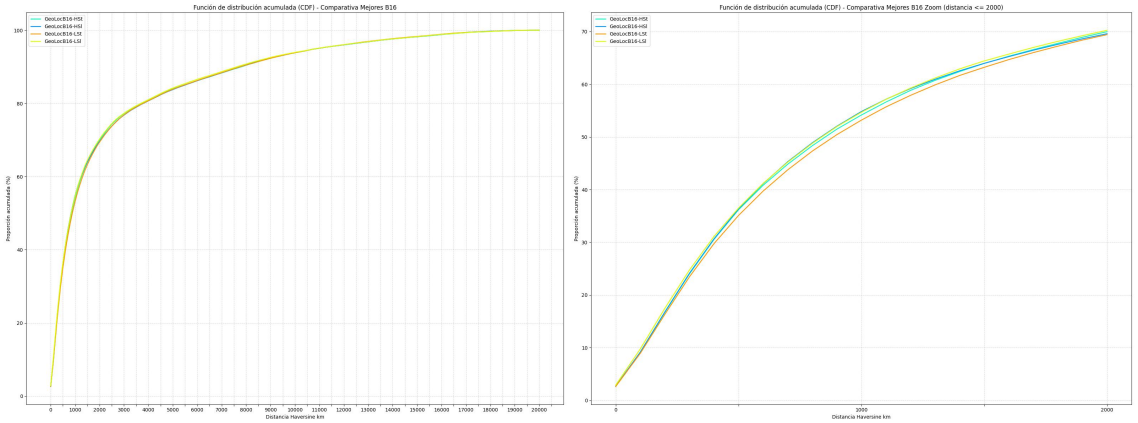

Figura 4.20: Mejores funciones de distribución acumulada de los modelos basados en `ViT-B/16` (ver imagen ampliada en el Apéndice B, Figura B.16)

Una de esas funciones es `InterCubicMonotonusHermiteLoss`, cuyos histogramas (véase el Apéndice B, Figuras B.5 y B.11) muestran una concentración de imágenes a la izquierda de la gráfica, con un pico menos agudo que en el resto de modelos, pero una base más ancha. Esto indica una dispersión mayor entre los valores cercanos al máximo que se da en las distancias pequeñas. Este comportamiento refleja a la perfección el enfoque de suficiencia tolerante planteado en la Sección 4.3.5.5 donde se pretendía diseñar unos modelos con una función de pérdida que valorase mucho el "estar suficientemente cerca", es decir, considerar como buenas todas aquellas predicciones dentro de un umbral de razonable cercanía (establecido en $4 \cdot \text{DMP} = 1748{,}92$ km). Por eso, el modelo no busca la estimación más precisa de una ubicación, sino garantizar que la mayoría de las predicciones se sitúen dentro de ese umbral.

Otras funciones destacables son `GeoscoreLoss` y `GeoscoreModificadoLoss` con su noción de penalizar con fuerza los errores cuando se está cerca del objetivo e ir suavizando gradualmente la pérdida. Todo bajo la idea de que es preferible estar "muy cerca" en vez de estar simplemente "menos lejos". Gracias a esa estrategia, los modelos entrenados con estas funciones consiguen al principio predecir con mayor

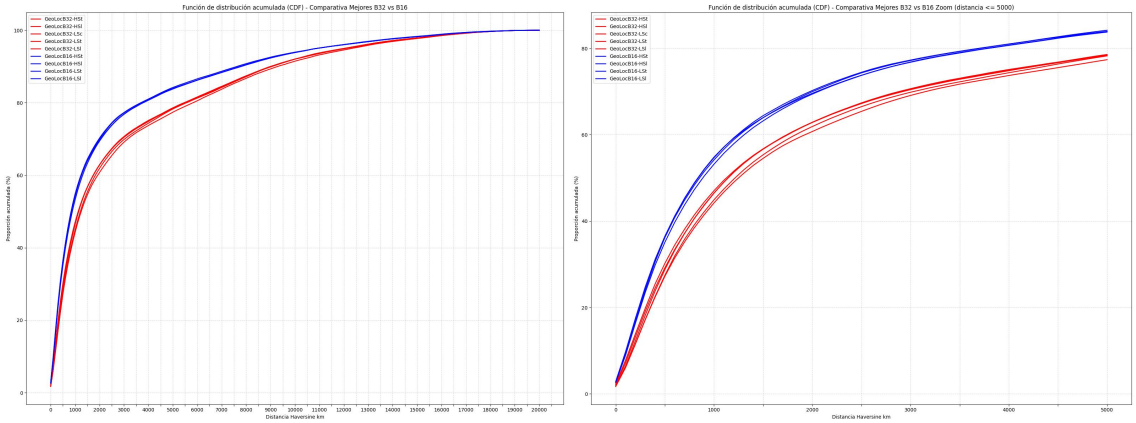

Figura 4.21: Comparación mejores funciones de distribución acumulada de los modelos B32 vs B16 (ver imagen ampliada en el Apéndice B, Figura B.17)

precisión un mayor número de imágenes en comparación con el resto. Sin embargo, a partir de cierto punto, su rendimiento baja frente a otros modelos ya que al enfocar el aprendizaje en corregir sobre todo los errores pequeños y reducir progresivamente la penalización, el modelo no se esfuerza tanto en mejorar las predicciones que quedan fuera de ese rango óptimo. Como resultado, cuando las distancias se vuelven más lejanas y superan ese umbral de "muy cerca", estas predicciones menos refinadas pierden competitividad frente a las de otros modelos que mantienen una penalización más uniforme. Se proporciona un análisis más detallado de lo mencionado en el Apéndice B, Sección B.3.

Los modelos entrenados con la función MSELoss no han destacado demasiado y eso tiene sentido, ya que se trataba de una primera aproximación al problema con una estrategia muy sencilla y que tenía carencias de precisión. Estas se debían a usar una métrica, que inicialmente está diseñada para espacios vectoriales en $\mathbb{R}^k$, para aproximar la naturaleza esférica de la superficie terrestre.

4.6. Desarrollo de la Aplicación

Se ha desarrollado una aplicación con interfaz gráfica, programada en *Python*, que requiere tener correctamente instaladas todas las dependencias necesarias para su funcionamiento. Una vez ejecutada, la interfaz resulta muy intuitiva, permitiendo al usuario cargar imágenes en formatos `*.jpg`, `*.png` o `*.jpeg` mediante el explorador de archivos. A partir de estas imágenes, el sistema realiza la predicción de su ubicación geográfica utilizando el mejor modelo identificado en la Sección 4.5.

En primer lugar, la imagen se prepara para ser procesada por el Codificador de Imágenes (*Image Encoder*). Una vez obtenido y normalizado el vector de características resultante, este se introduce en el Modelo de Geolocalización, que devuelve como salida las coordenadas predichas (latitud y longitud). Estas coordenadas se muestran tanto en la interfaz gráfica como en la consola.

A continuación, se genera un mapa centrado en la ubicación estimada, marcando dicha posición. Este mapa se guarda en formato `.html` y se abre automáticamente

en el navegador web, utilizando la librería `Folium` (Folium Developers, 2025) y la librería `webbrowser`. En la Figura 4.22 se puede observar un ejemplo del mapa generado con la ubicación predicha.

El usuario puede seguir cargando imágenes y realizando nuevas predicciones de forma continua hasta cerrar la aplicación. En la Figura 4.23 se muestra un ejemplo de la interfaz gráfica desarrollada.

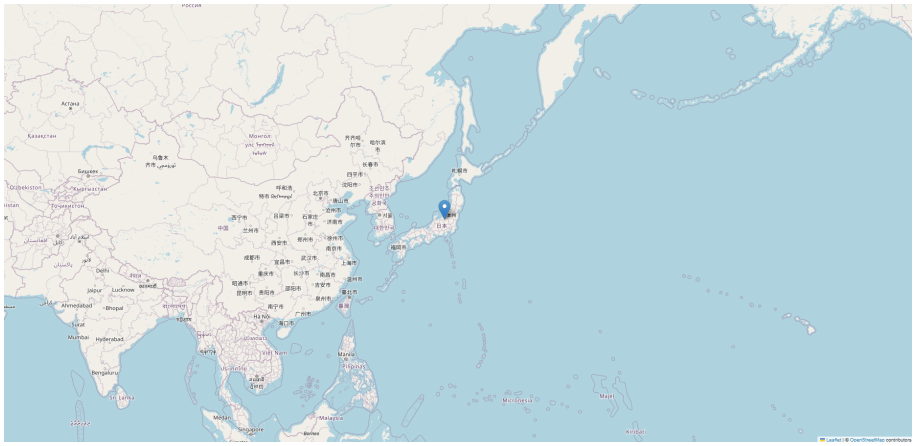

Figura 4.22: Mapa Mundi con la ubicación de la imagen predicha

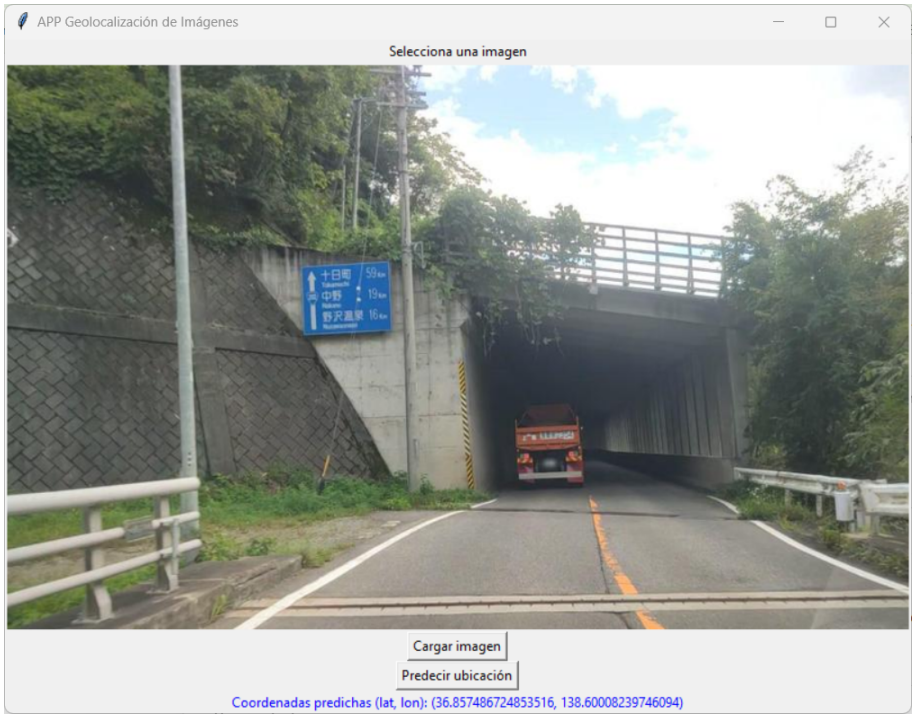

Figura 4.23: Muestra de ejemplo de la Interfaz de la Aplicación

Capítulo 5

Conclusiones y Trabajo Futuro

En el presente trabajo se propuso diseñar un modelo de aprendizaje automático, lo más robusto y preciso posible, que fuese capaz de estimar la ubicación geográfica de una imagen, apoyándose únicamente en características visuales presentes en ella, sin recurrir a información extra.

Asimismo, se planteó la necesidad de encontrar un extenso conjunto de datos etiquetados que proporcionara numerosos ejemplos para entrenar un modelo basado en aprendizaje supervisado. Además, este conjunto debía garantizar a la vez una distribución geográfica equilibrada libre de sesgos, como su localizabilidad, es decir, que la mayoría de las imágenes incluyesen la información necesaria para realizar estimaciones precisas de su ubicación.

Por último, se diseñó una aplicación donde se pudiese utilizar el modelo entrenado para realizar predicciones subiendo imágenes a través de una interfaz sencilla.

La construcción del modelo, la obtención del conjunto de datos y la creación de la aplicación han sido logrados con éxito. A continuación, se describirán las conclusiones del proyecto y algunos posibles trabajos futuros que podrían continuar desarrollándose como resultado de la presente investigación.

5.1. Conclusiones

Para alcanzar todos los objetivos planteados en este proyecto, fue necesario superar seis fases fundamentales.

En la primera fase, se seleccionó un conjunto de datos amplio y diverso, adecuado para entrenar un modelo de geolocalización de imágenes. Se justificó su elección en base a su riqueza geográfica y su buena distribución equitativa. También se logró una manera eficiente de almacenar y gestionar semejante volumen de datos para las siguientes fases.

En la segunda fase, se llevó a cabo la división del conjunto de datos en tres partes: entrenamiento, validación y test. Esta separación permitió, respectivamente, entrenar el modelo, determinar el número óptimo de épocas que se mantuvo entrenando y realizar una evaluación rigurosa del rendimiento final.

A continuación, se definieron los parámetros y arquitecturas de los distintos

modelos que se iban a desarrollar, explorando diferentes estrategias con el fin de identificar cuál ofrecía el mejor comportamiento frente al problema de geolocalización.

En la cuarta fase, se entrenaron los modelos definidos, monitorizando todo el proceso para evitar tanto el *overfitting* como el *underfitting* y así obtener un modelo con buen equilibrio entre aprendizaje y generalización.

Posteriormente, se evaluaron todos los modelos entrenados para analizar todas las gráficas y tomar decisiones informadas para seleccionar el modelo con el mejor desempeño entre precisión y capacidad de generalización frente a datos no vistos.

Los resultados obtenidos fueron satisfactorios: el modelo final logró mantener un excelente equilibrio entre precisión y consistencia, logrando que más del 50 % de las imágenes de evaluación fuesen predichas a menos de 900 km de su ubicación real.

En la última fase, se implementó la aplicación mencionada, que, gracias a una interfaz sencilla e intuitiva, permite a cualquier usuario utilizar el modelo subiendo imágenes y obteniendo estimaciones de su ubicación.

Este proyecto y sus resultados han demostrado que las redes neuronales son una herramienta sumamente eficaz para el problema de la geolocalización de imágenes. Su capacidad para generar con sus neuronas representaciones y relaciones complejas, junto con su habilidad de generalizar con éxito resultados ante datos nuevos, las convierte en una opción especialmente prometedora. Más aún sabiendo la facilidad que habría en internet de conseguir un conjunto de datos más extenso y rico que el usado si la mayoría de *datasets* con imágenes geolocalizadas no fueran privados.

Las principales limitaciones encontradas durante el desarrollo han sido computacionales y de tiempo. Por un lado, el manejo eficiente de una cantidad masiva de datos representó un reto considerable. Por otro lado, la limitada capacidad de cómputo de los equipos que se disponía impidió explorar modelos más complejos con mayor capacidad de representación. Y es que impidió mejoras en el codificador de imágenes (*Image Encoder*), un aumento del tamaño del modelo o una búsqueda más exhaustiva de hiperparámetros que podrían haber conducido a resultados aún más precisos. Para mitigar parcialmente esta limitación de tiempo, se optó por utilizar dos ordenadores para poder entrenar varios modelos a la vez y, al mismo tiempo, generar los *embeddings* del conjunto de datos.

5.2. Trabajo Futuro

A raíz de este trabajo de fin de grado, se desbloquean diversas líneas de investigación que quedan abiertas para explorarse en el futuro. Durante la elaboración del proyecto han surgido algunas ideas y preguntas que no pudieron tratarse en el momento y han quedado pendientes de ser abordadas próximamente. Algunas de estas propuestas están directamente relacionadas con lo implementado y son extensiones naturales del trabajo actual; sin embargo, no se pudieron desarrollar por limitaciones de tiempo, alcance o capacidad computacional disponible. Otras, en cambio, son más generales y pueden ser buenas exploraciones complementarias para otros investigadores para que amplíen el campo de estudio.

A continuación se van a presentar algunos trabajos futuros, tanto para mejorar

el modelo actual como para explorar nuevos enfoques:

- Profundizar más en la búsqueda de valores óptimos de los parámetros actualmente fijados, como puede ser el valor de paciencia o el tamaño de la red. También se podrían probar diferentes configuraciones experimentando con otros optimizadores, distintas combinaciones de funciones de activación o la aplicación de *dropout* y de alguna otra técnica de regularización.
- Explorar como mejora el rendimiento del modelo utilizando los dos mejores codificadores de imagen de `CLIP`, que son `ViT-L/14` y `ViT-L/14-336px`. Su mejor capacidad de representación podría aportar mejoras significativas.
- Desarrollar una versión web o móvil más avanzada de la aplicación actual, que tenga funcionalidades adicionales como la selección entre varios modelos o un modo de juego en el que el usuario pueda competir contra la IA utilizando imágenes precargadas del conjunto de test.
- Explorar una posible mejora del modelo actual, en la que múltiples redes realicen predicciones de forma paralela y se seleccione la ubicación más probable a partir de la región donde más se acierte.
- Construir un nuevo conjunto de datos más amplio, equilibrado y representativo que supere las limitaciones del conjunto de entrenamiento actual y sea más extenso y diverso que el conjunto de test para poder desarrollar modelos más completos.
- Desarrollar un codificador de imágenes personalizado que aprenda un espacio de representación diseñado específicamente para la tarea de geolocalización del que luego se entrenen módulos de geolocalización.
- Dividir el módulo de geolocalización en dos submodelos conectados de manera secuencial para crear una nueva arquitectura donde el primer modelo se encarga de identificar el país al que pertenece la imagen, y el segundo, se especializa en estimar una ubicación precisa dentro del país identificado. Esta división permitiría aprovechar modelos regionales más precisos y especializados.
- Analizar el rendimiento del modelo actual en otros campos como puede ser la identificación del lenguaje, viendo la ubicación predicha y escogiendo el idioma principal de esa región.

Chapter 6

Introduction

6.1. Motivation

Determining the precise location of an image while taking into consideration all the Earth's surface is a complex challenge in the field of computer vision algorithms. For that reason, it is particularly relevant for evaluating and developing classification algorithms, specifically those based on Machine Learning, such as neural networks.

Addressing this problem offers me an excellent opportunity to apply all the concepts that I have learnt in the Artificial Intelligence (AI) subjects, in particular my knowledge of neural networks and classifiers. Moreover, whether by pursuing a master's degree or focusing on my future career path, this project aligns with my long-term aspirations in the AI field.

Despite its potential, few supervised learning approaches are actually trained for image geolocation tasks. This was attributed until recently to two limitations that most geolocated image datasets had: (1) a strong geographic bias due to an overrepresentation of certain regions and a lack of diversity, making global scale geolocation unapproachable; (2) a high percentage of the tagged images being non-localizable because of their low quality or the absence of distinctive elements that allow them to be geographically differentiated.

Nevertheless, the recently published `OpenStreetView-5M` (Astruc et al., 2024) dataset greatly mitigates the aforementioned limitations, making it an ideal resource for the present study. This is an open-access, large-scale dataset containing more than 5.1 million geolocated street view images covering a total of 225 countries and territories.

In addition, image geolocation has numerous applications in different fields. For example, *GPS-free location*, which is particularly useful if someone is lost and must determine their precise position when GPS signals are unavailable or unreliable; *journalistic fact-checking* to validate whether the images were taken at the location claimed; and *criminal investigation support* by allowing image location tracking, which can aid in the identification of suspects or victims.

6.2. Objectives

The main objective of this project is to develop a supervised Machine Learning model that is as robust as possible, capable of precisely geolocating images and able to adapt well to new geographic environments. To this end, a series of benchmarks will be designed to evaluate and compare different alternatives, identifying the best approaches to the problem.

What is really being sought is a model that learns to identify key geographic features instead of making predictions based on superficial patterns or overfitting and memorization of the training dataset. Some of these characteristics might include, for example, traffic signs, architectural styles, or natural elements such as weather or vegetation.

Therefore, because what is really intended is to design a model with a deeper and more generalizable understanding of relevant geographic aspects, it is essential to have a training set that avoids biases resulting from overrepresented Earth's regions. That's where the already mentioned `OpenStreetView-5M` comes in, a dataset that provides a standardized and reliable base. This will be used to train and test the model, minimizing bias risk and supporting generalization to new images.

Finally, an application will be developed to use the already trained model to make image location predictions. This application will allow the user to upload local images to obtain coordinates that estimate their geographic position. Furthermore, it will have a simple interface to facilitate interaction between the model and the user.

6.3. Work Plan

The following section presents the work plan that will be carried out to achieve all the objectives previously outlined.

1. *Dataset Selection and Preparation.* First, the selection of the `OpenStreetView-5M` dataset will be analyzed and justified. Once the dataset has been selected, an efficient way to store and manage such a large dataset will be investigated. This organization will be essential for easily accessing and processing the data in the following phases of the project. *(Estimated duration: 27/01/2025 - 09/02/2025, 2 weeks)*

2. *Dataset Splitting.* The dataset will be divided into three parts (training, validation and testing). This separation is necessary to avoid overfitting, to determine the best parameters configurations and to ensure a reliable final evaluation of the models' performance. *(Estimated duration: 10/02/2025 - 16/02/2025, 1 week)*

3. *Definition of the Models and Their Parameters.* The architectures of the models will be defined, along with the parameters and options available for their development. Thus, different configurations and strategies will be tested to find

the one that offers the best performance. *(Estimated duration: 17/02/2025 - 09/03/2025, 3 weeks)*

4. *Models Training.* Once all the models and their corresponding parameters have been defined, the training stage will be carried out. During this phase, the hyperparameters will be adjusted and the entire training and validation process will be monitored. *(Estimated duration: 10/03/2025 - 13/04/2025, 5 weeks)*

5. *Performance Evaluation and Best Model Selection.* All the different models will be evaluated against the test set after the training stage. Afterwards, the evaluation graphs will be analyzed along with the rest of the generated data to compare their performance and make informed decisions. Based on the results, the model with the best performance in terms of accuracy and ability to generalize to new data will be selected. This model will be used in the last phase of the project. *(Estimated duration: 14/04/2025 - 27/04/2025, 2 weeks)*

6. *Application Development.* Finally, the aforementioned application will be developed with a simple and easy to use interface so that the user can use the model selected in the previous phase. This interface will allow users to upload new images to obtain their coordinates predicted by the model. *(Estimated duration: 28/04/2025 - 04/05/2025, 1 week)*

Chapter 7

Conclusions and Future Work

In this work, we proposed the design of a robust and accurate Machine Learning model capable of estimating the geographical location of an image based solely on its visual features, without resorting to any extra information.

Likewise, we addressed the need to find a large geotagged dataset with numerous examples for training a neural network. In addition, this dataset had to guarantee a balanced geographic distribution free of biases, as well as localizability, meaning that most images should include enough distinctive information to make accurate location estimation.

Finally, an application was developed to enable the trained model to make predictions by uploading images through a simple interface.

The model's construction, the dataset's acquisition, and the application's development were successfully completed. Subsequently, the project's conclusions and possible future work will be discussed.

7.1. Conclusions

In order to accomplish all the objectives presented in this work, it was necessary to go through six fundamental phases.

In the first phase, a large and varied dataset was carefully selected for training an image geolocation model. This choice was justified based on its geographic diversity and even distribution. An efficient way to store and manage such a large volume of data for subsequent phases was also achieved.

In the second phase, the dataset was divided into three parts: train, validation, and test. This partition allowed respectively, the training of the models, the determination of the optimal number of epochs to continue training, and the rigorous evaluation of the final performance.

Thereafter, the parameters and architectures of the different models were defined, exploring various strategies to identify which offered the best performance for the geolocation problem.

In the fourth phase, the defined models were trained, with the entire process being monitored to avoid both overfitting and underfitting, thus obtaining a well-

balanced model between learning and generalization.

Subsequently, all trained models were evaluated. The evaluation graphs, along with the rest of the generated data, were analyzed to select the model with the best performance in terms of accuracy and generalization ability against unseen data.

The results obtained met expectations: the final model managed to maintain an excellent balance between accuracy and consistency, achieving predictions with errors below 900 km in more than 50% of the test images.

In the last phase, the aforementioned application was implemented with a simple and intuitive interface that allows any user to use the model by uploading images and obtaining their location estimations.

This project and its results have demonstrated that neural networks are an extremely effective tool for the image geolocation problem. Their capacity to generate complex representations and relationships, along with their ability to successfully generalize results to unseen data, makes them a particularly promising option. Even more so considering how easy it would be to access a larger and richer dataset from the internet if the majority of geotagged datasets were not private.

The main limitations encountered during the project development were computational and time-related. On the one hand, the efficient management of a massive amount of data represented a considerable challenge. On the other hand, the computational limitations of the available equipment hindered the exploration of more complex models with greater representational capacity. This prevented improvements to the image encoder, increasing the model size, or conducting a more exhaustive hyperparameter search that could have led to even more accurate results. To partially mitigate the time limitation, it was decided to use two computers to train several models or generate the dataset embeddings simultaneously.

7.2. Future Work

As a result of this final degree project, several lines of research have been opened for future exploration. During the development of this work, some ideas and questions arose but could not be addressed at the time, so they remain pending for future work. Some of these proposals are directly related to what has already been implemented and are natural extensions of the current work. Nevertheless, they could not be developed due to time, scope, and computational limitations. Others, however, are more general and may serve as valuable complementary studies for other researchers aiming to expand the field.

Subsequently, some possible future work directions will be presented, both to improve the current model and to explore new approaches:

- Deepen the search for optimal values for the current set of parameters, such as patience value or neural network size. Moreover, different configurations could be tested by experimenting with other optimizers, combinations of activation functions, and the application of dropout or other regularization techniques.

- Explore how the performance of the model improves when using the two best

`CLIP` image encoders (`ViT-L/14` and `ViT-L/14-336px`), whose enhanced representation capacity could lead to significant improvements.

- Develop a more advanced web or mobile version of the current application, incorporating additional features, such as model selection options or a game mode in which the user competes against the AI using preloaded images from the test set.
- Explore a potential improvement to the current model in which multiple networks make parallel predictions, and the most likely location is selected based on the region with the highest number of correct predictions.
- Build a new, larger, more balanced, and more representative dataset to overcome the limitations of the current training set, along with a more extensive and diverse test set to support the development of more comprehensive models.
- Develop a custom image encoder that learns a new representation space specifically designed for the geolocation task, from which geolocation modules can then be trained.
- Split the geolocation module into two sequentially connected submodels to create a new architecture in which the first model identifies the country of an image and the second one specializes in estimating the location using both the image and the predicted country. This separation would allow the use of more accurate and specialized regional models.
- Analyze the performance of the current model in other areas, such as language identification, by inferring the language of an image based on the most commonly spoken language in the predicted location.

Bibliografía

Lo que sabemos es una gota de agua; lo que ignoramos es el océano

Isaac Newton

Astruc, G., Dufour, N., Siglidis, I., Aronssohn, C., Bouia, N., Fu, S., Loiseau, R., Nguyen, V. N., Raude, C., Vincent, E., Xu, L., Zhou, H. y Landrieu, L. OpenStreetView-5M: The many roads to global visual geolocation. *CVPR*, 2024. https://github.com/gastruc/osv5m.

Awasthi, P., Dikkala, N., Kamath, P. y Meka, R. Learning neural networks with sparse activations. En *Proceedings of Thirty Seventh Conference on Learning Theory* (editado por S. Agrawal y A. Roth), vol. 247 de *Proceedings of Machine Learning Research*, páginas 406–425. PMLR, 2024.

Bergmann, D. y Stryker, C. What is a loss function? https://www.ibm.com/think/topics/loss-function, 2025. Accedido: 24-05-2025.

Brummelen, G. V. *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton University Press, 2013. ISBN 9780691148922.

Bürkner, P.-C., Gabry, J., Kay, M. y Vehtari, A. posterior: Tools for working with posterior distributions. https://mc-stan.org/posterior/reference/entropy.html, 2024. R package version 1.6.0 Consultado en la función `entropy` Accedido: 24-05-2025.

Cheng, G., Han, J. y Lu, X. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, vol. 105(10), páginas 1865–1883, 2017. Doi: 10.1109/JPROC.2017.2675998.

Datacolor. ¿Qué es el CIELAB? https://www.datacolor.com/es/business-solutions/blog/que-es-cielab/, 2023. Accedido: 24-05-2025.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S. et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Folium Developers. Folium. https://github.com/python-visualization/folium, 2025. Accedido: 19-04-2025.

García, S., Luengo, J. y Herrera, F. *Data Preprocessing in Data Mining*, vol. 72 de *Intelligent Systems Reference Library*. Springer International Publishing, Cham, 2015. ISBN 978-3-319-10246-7 978-3-319-10247-4. Doi: 10.1007/978-3-319-10247-4.

GeeksforGeeks. Ml | underfitting and overfitting. https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/, 2025. Accedido: 07-05-2025.

Geiger, A., Lenz, P. y Urtasun, R. Are we ready for autonomous driving? the kitti vision benchmark suite. En *2012 IEEE conference on computer vision and pattern recognition*, páginas 3354–3361. IEEE, 2012.

GeoGuessr. Geoguessr. https://www.geoguessr.com/es, 2025. Accedido: 24-05-2025.

GeoTips. Tips and tricks for geoguessr game - africa. https://geotips.net/africa/, 2025. Accedido: 24-05-2025.

González, A. G. Normalización de datos en machine learning: Métodos y aplicaciones. https://panamahitek.com/normalizacion-de-datos-en-machine-learning-metodos-y-aplicaciones/, 2023. Accedido: 08-05-2025.

Google. Google street view. https://www.google.com/streetview/, 2024. Accedido: 11-05-2025.

Hays, J. y Efros, A. A. Im2gps: estimating geographic information from a single image. En *2008 IEEE Conference on Computer Vision and Pattern Recognition*, páginas 1–8. 2008. Doi: 10.1109/CVPR.2008.4587784.

He, K., Zhang, X., Ren, S. y Sun, J. Deep residual learning for image recognition. En *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, páginas 770–778. 2016. Doi: 10.1109/CVPR.2016.90.

Helber, P., Bischke, B., Dengel, A. y Borth, D. EuroSAT: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12(7), páginas 2217–2226, 2019. Doi: 10.1109/JSTARS.2019.2918242.

Hugging Face. Hugging face datasets documentation. https://huggingface.co/docs/datasets/index, 2025. Accedido: 24-05-2025.

Interactive Chaos. Tasa de aprendizaje - tutorial de deep learning. https://interactivechaos.com/es/manual/tutorial-de-deep-learning/tasa-de-aprendizaje, 2025. Accedido: 24-05-2025.

Izbicki, M., Papalexakis, E. E. y Tsotras, V. J. Exploiting the earth's spherical geometry to geolocate images. En *Machine Learning and Knowledge Discovery in Databases* (editado por U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis y C. Robardet), páginas 3–19. Springer International Publishing, Cham, 2020. ISBN 978-3-030-46147-8.

Kingma, D. P. y Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Knott, G. D. *Interpolating cubic splines*, vol. 18. Springer Science & Business Media, 1999.

Kornblith, S., Shlens, J. y Le, Q. V. Do better imagenet models transfer better? En *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, páginas 2661–2671. 2019.

Manuel, J. R. Práctica de coordenadas geográficas. https://es.slideshare.net/slideshow/practica-de-coordenadas-geogrficas/66910934, 2016. Accedido: 18-04-2025.

Mapillary. Mapillary. https://www.mapillary.com, 2025. Accedido: 24-05-2025.

Matos, A. Rest countries api documentation. https://restcountries.com/#api-endpoints-using-this-project, 2025. Inspirado por restcountries.eu de Fayder Florez. Accedido: 24-05-2025.

Murel, J. y Noble, J. What is an encoder-decoder model? https://www.ibm.com/think/topics/encoder-decoder-model, 2024. Accedido: 08-05-2025.

Plonkit Team. Beginner's guide to geoguessr. https://www.plonkit.net/beginners-guide, 2025. Accedido: 15-04-2025.

Popescu, M.-C., Balas, V. E., Perescu-Popescu, L. y Mastorakis, N. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, vol. 8(7), páginas 579–588, 2009.

Proyecto Viajero. Coordenadas geográficas: Latitud y longitud. https://proyectoviajero.com/coordenadas-geo-latitud-longitud/, 2024. Accedido: 18-04-2025.

PyTorch. Pytorch documentation. https://pytorch.org/docs/stable/, 2025. Accedido: 24-05-2025.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J. et al. Learning transferable visual models from natural language supervision. En *International conference on machine learning*, páginas 8748–8763. PMLR, 2021.

Samet, H. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, vol. 16(2), páginas 187–260, 1984. Doi: 10.1145/356924.356930.

Singh, D. Overfitting, underfitting, and bias-variance tradeoff. https://medium.com/geekculture/overfitting-underfitting-and-bias-variance-tradeoff-9e83f4a147c, 2021. Publicado en Geek Culture. Accedido: 18-04-2025.

Spahić, A. Geoguessr hack/exploit with proxy (fiddler on windows). https://amelspahic.com/geoguessr-exploit-with-proxy-fiddler-on-windows, 2022. Accedido: 11-05-2025.

Symas Corporation. Lightning memory-mapped database. https://www.symas.com/mdb, 2025. Accedido: 24-05-2025.

The OpenLDAP Project. OpenLDAP Software: open source implementation of the Lightweight Directory Access Protocol. https://www.openldap.org/, 2025. Accedido: 24-05-2025.

Thomee, B., Shamma, D. A., Friedland, G., Elizalde, B., Ni, K., Poland, D., Borth, D. y Li, L.-J. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, vol. 59(2), páginas 64–73, 2016.

Ultralytics. Tamaño del lote en el aprendizaje profundo. https://www.ultralytics.com/es/glossary/batch-size, 2025. Accedido: 24-05-2025.

Vo, N., Jacobs, N. y Hays, J. Revisiting im2gps in the deep learning era. En *Proceedings of the IEEE international conference on computer vision*, páginas 2621–2630. 2017.

Watson, J. N. py-lmdb: Python bindings for lmdb. https://github.com/jnwatson/py-lmdb, 2025. Accedido: 24-05-2025.

Wikipedia. Interpolación cúbica monótona. https://es.wikipedia.org/wiki/Interpolaci%C3%B3n_c%C3%BAbica_mon%C3%B3tona, 2025a. Accedido: 15-04-2025.

Wikipedia. Iso 3166. https://es.wikipedia.org/wiki/ISO_3166, 2025b. Accedido: 13-04-2025.

Ying, X. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, vol. 1168(2), página 022022, 2019. Doi: 10.1088/1742-6596/1168/2/022022.

Apéndice A

Gráficas de Evolución de Error durante Entrenamiento

Este apéndice tiene como objetivo proporcionar material suplementario que documenta el comportamiento de los modelos durante su fase de entrenamiento. En particular, se exponen todas las gráficas de evolución del error durante el entrenamiento a lo largo de las épocas. Estas gráficas presentan en color azul la media del error en cada época del conjunto de entrenamiento y en rojo la medida del error del conjunto de validación. Asimismo, estas gráficas permiten analizar la convergencia del proceso de entrenamiento y detectar posibles problemas como sobreajuste o subajuste.

A.1. Modelos CLIP-ViT-B/32 y CLIP-ViT-B/16

A continuación, se presentan todas las representaciones gráficas correspondientes a los modelos, tanto aquellos que utilizan como *Image Encoder* la arquitectura `CLIP-ViT-B/32` como los de `CLIP-ViT-B/16`. Estas visualizaciones se organizarán a su vez en subsecciones diferenciadas según esa arquitectura utilizada y la métrica empleada durante el entrenamiento de cada modelo.

A.1.1. B32-MSELoss

Hay tres modelos: `GeoLocB32-MSEc`, `GeoLocB32-MSEt` y `GeoLocB32-MSEl`

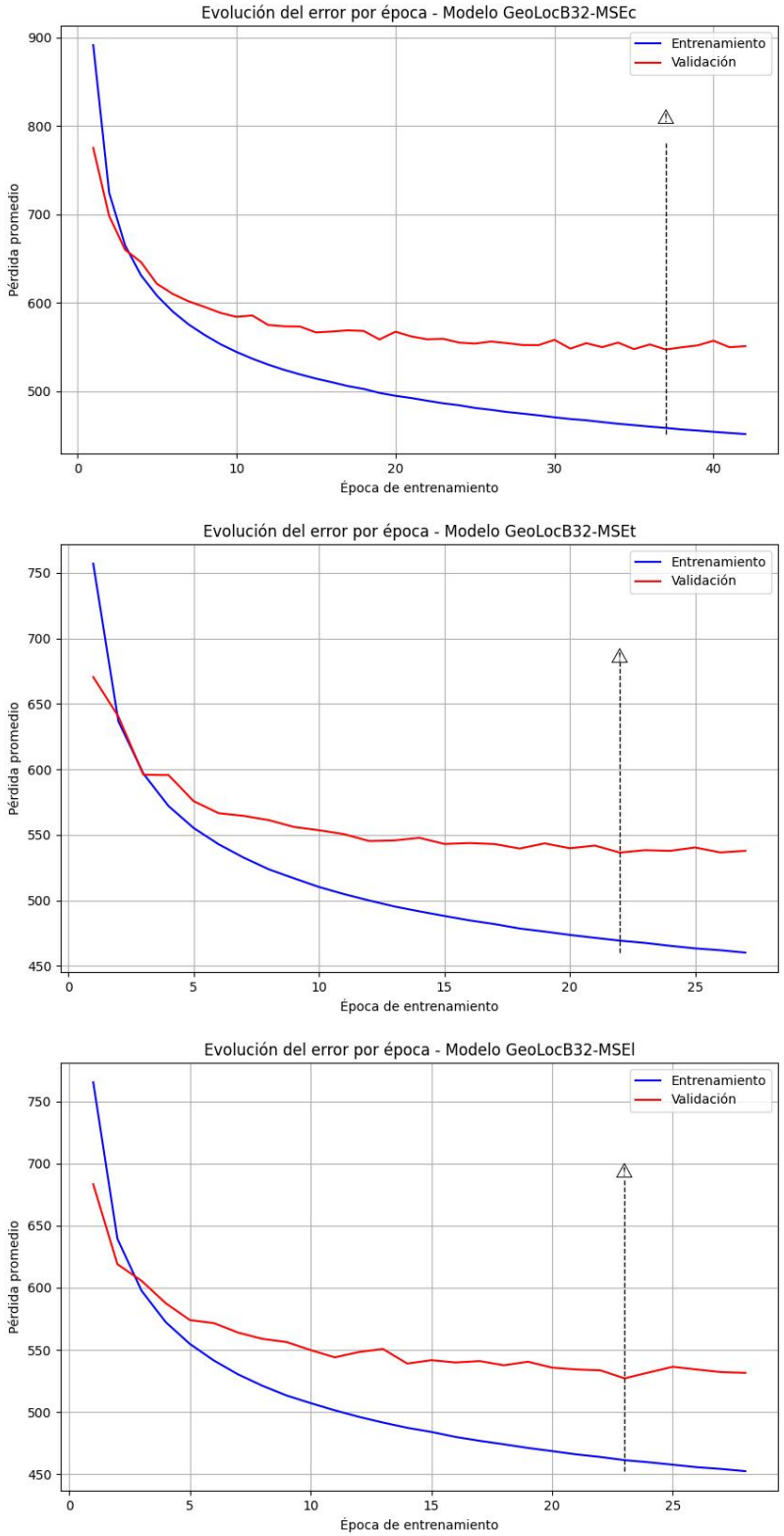

Figura A.1: Evolución de error - Modelos `GeoLocB32-MSE`

A.1.2. B32-HaversineLoss

Hay tres modelos: `GeoLocB32-HSc`, `GeoLocB32-HSt` y `GeoLocB32-HSl`

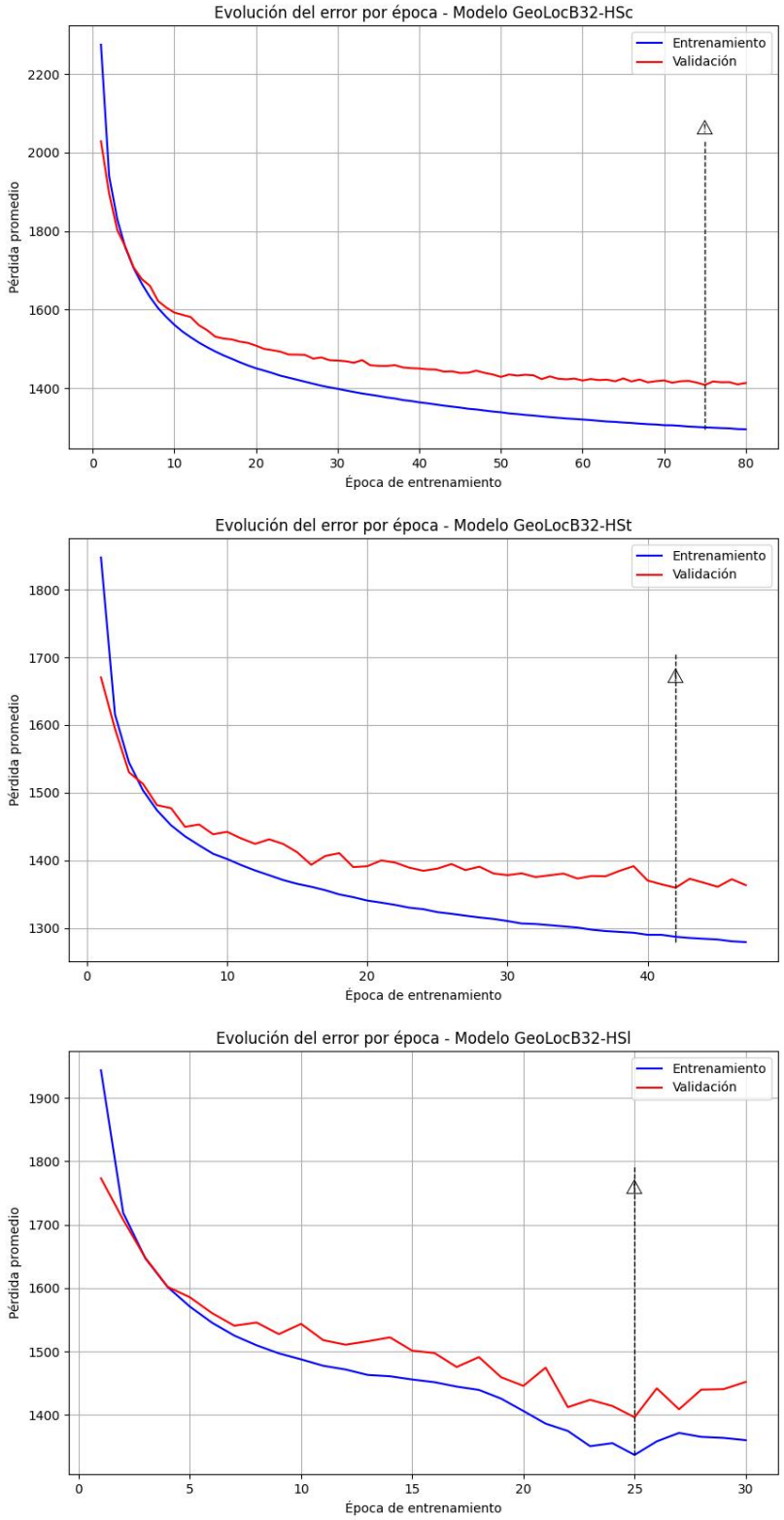

Figura A.2: Evolución de error - Modelos `GeoLocB32-HS`

A.1.3. B32-GeoscoreLoss

Hay tres modelos: `GeoLocB32-GSc`, `GeoLocB32-GSt` y `GeoLocB32-GSl`

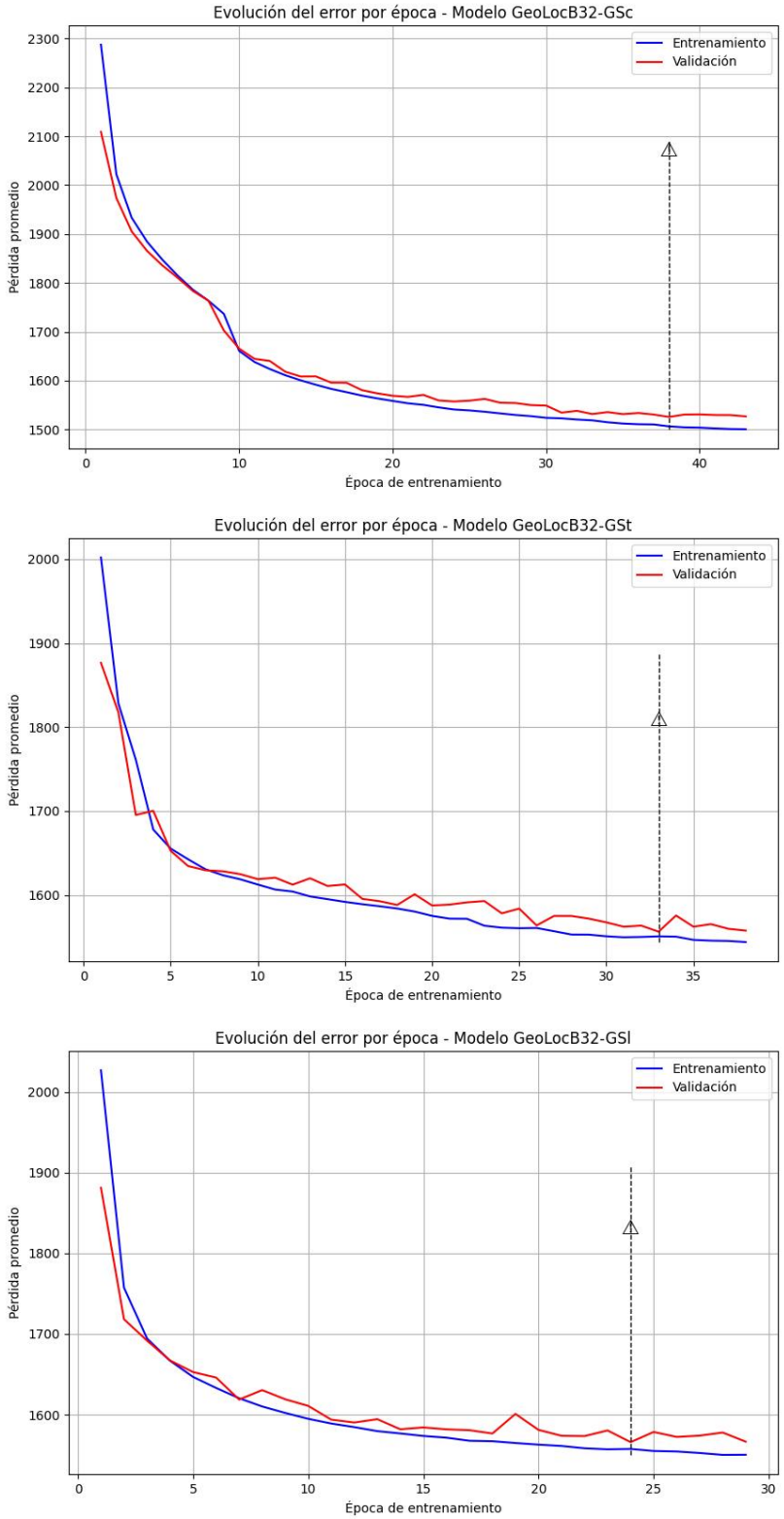

Figura A.3: Evolución de error - Modelos `GeoLocB32-GS`

A.1.4. B32-LinealScoreLoss

Hay tres modelos: `GeoLocB32-LSc`, `GeoLocB32-LSt` y `GeoLocB32-LSl`

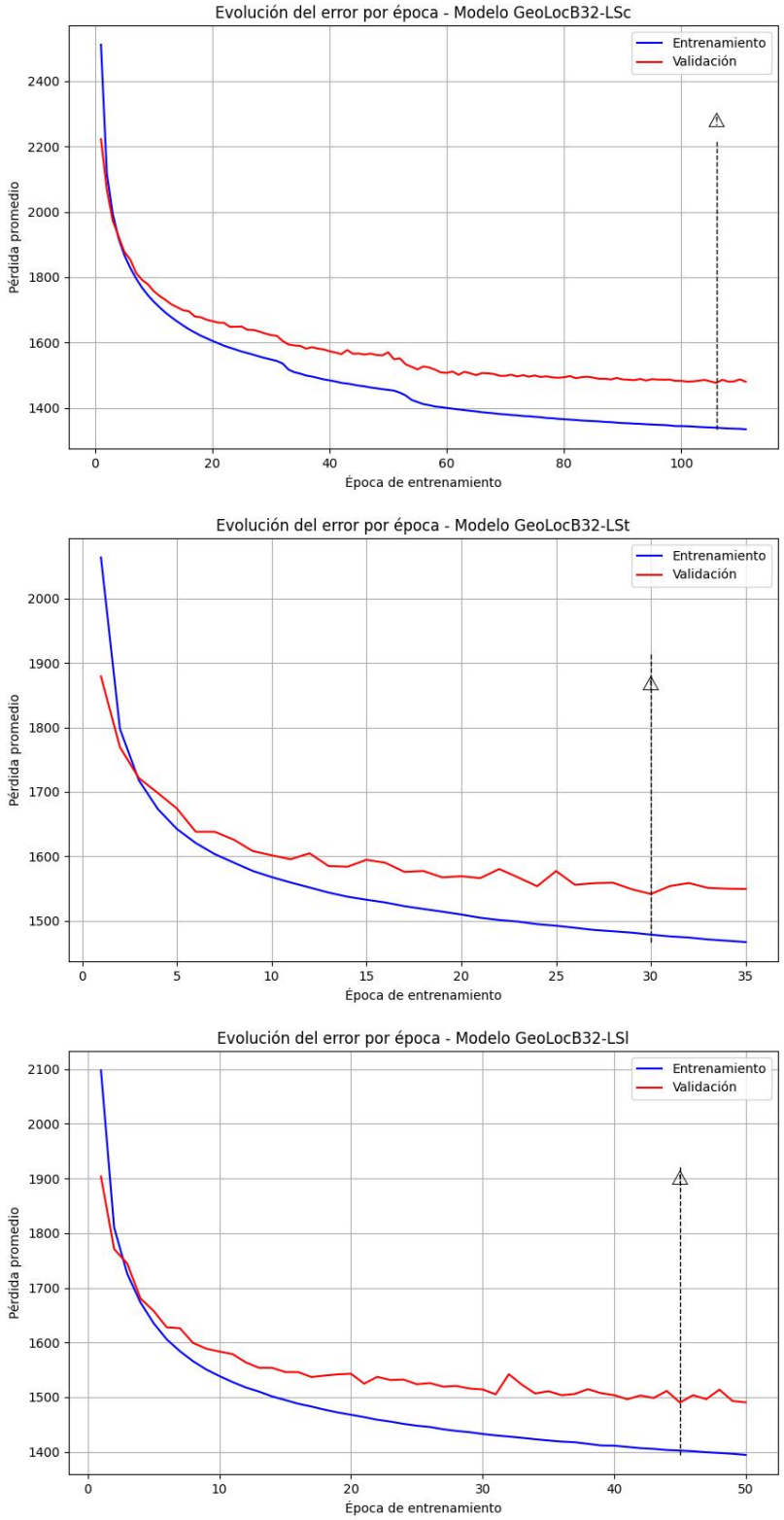

Figura A.4: Evolución de error - Modelos `GeoLocB32-LS`

A.1.5. B32-InterCubicMonotonusHermiteLoss

Hay tres modelos: `GeoLocB32-ICMHc`, `GeoLocB32-ICMHt` y `GeoLocB32-ICMHl`

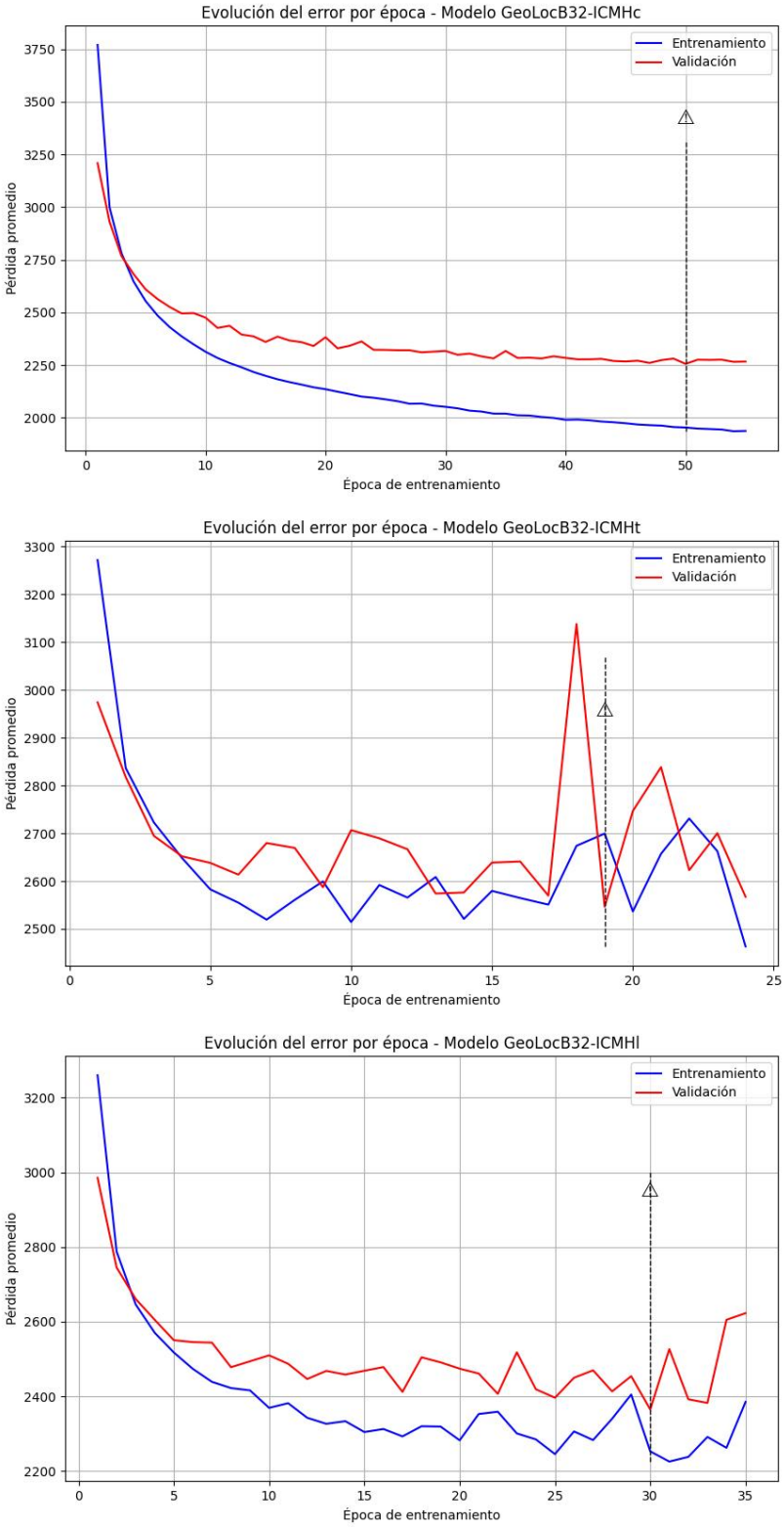

Figura A.5: Evolución de error - Modelos `GeoLocB32-ICMH`

A.1.6. B32-GeoscoreModificadoLoss

Hay tres modelos: `GeoLocB32-GSMc`, `GeoLocB32-GSMt` y `GeoLocB32-GSMl`

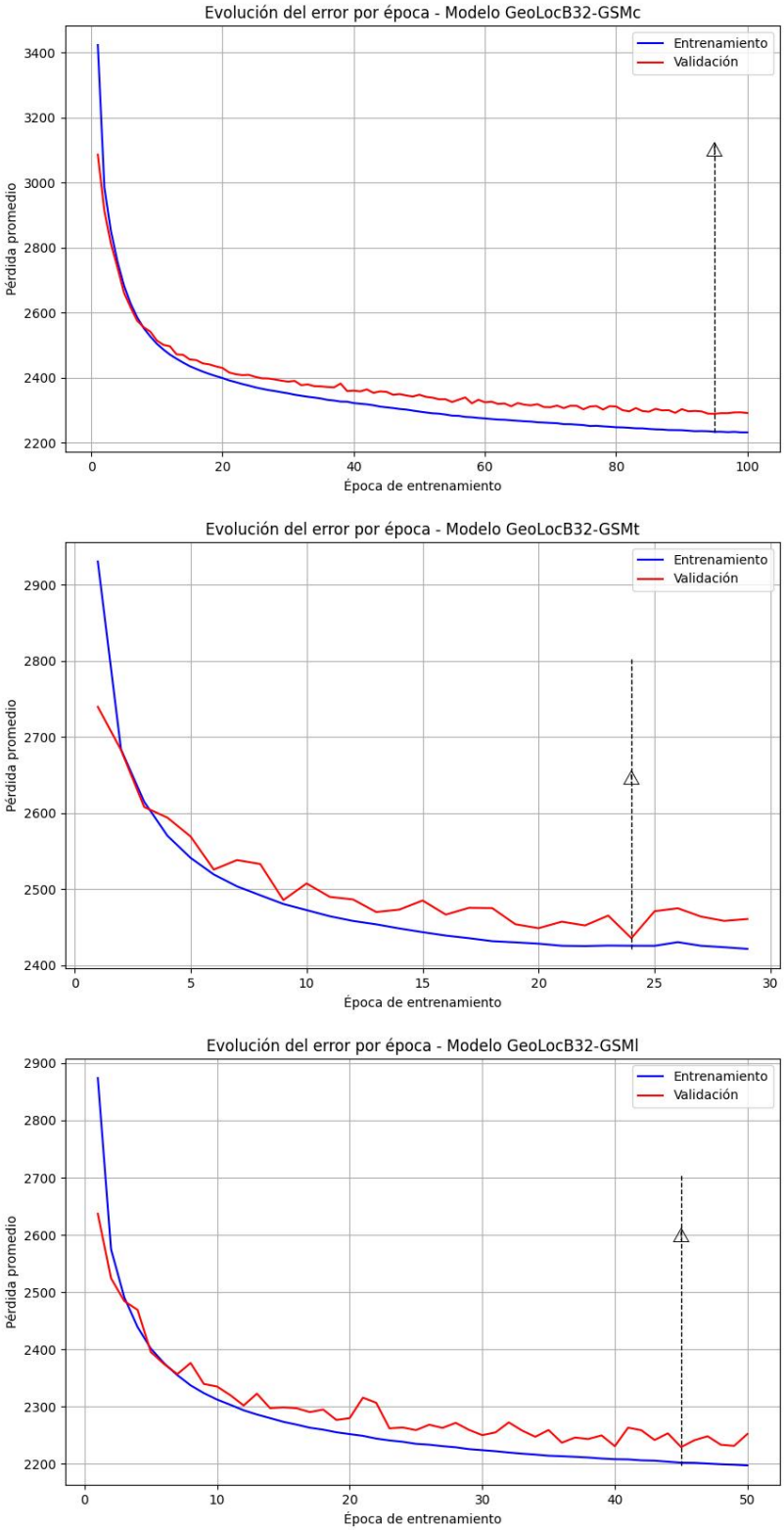

Figura A.6: Evolución de error - Modelos `GeoLocB32-GSM`

A.1.7. B16-MSELoss

Hay tres modelos: `GeoLocB16-MSEc`, `GeoLocB16-MSEt` y `GeoLocB16-MSEl`

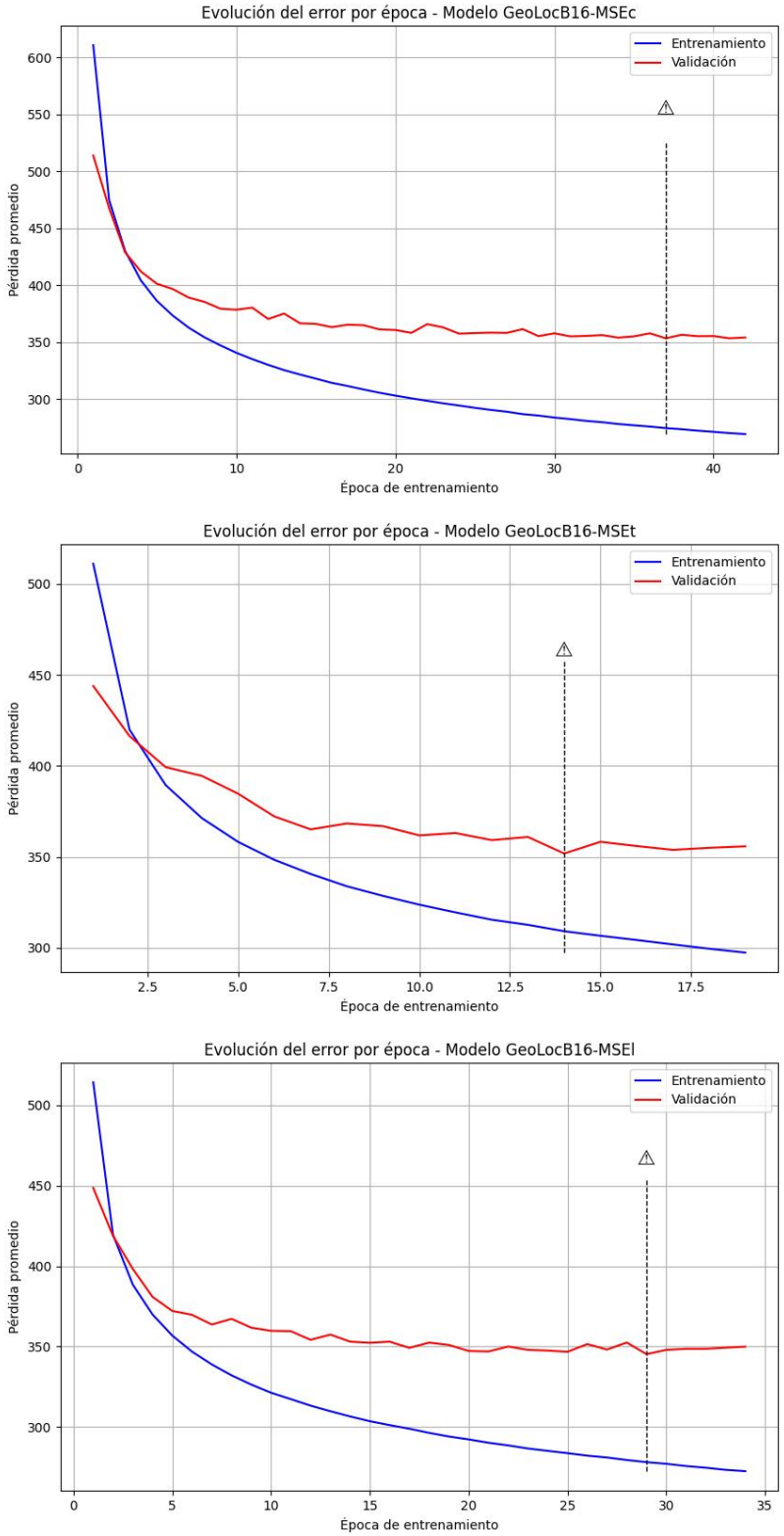

Figura A.7: Evolución de error - Modelos `GeoLocB16-MSE`

A.1.8. B16-HaversineLoss

Hay tres modelos: `GeoLocB16-HSc`, `GeoLocB16-HSt` y `GeoLocB16-HSl`

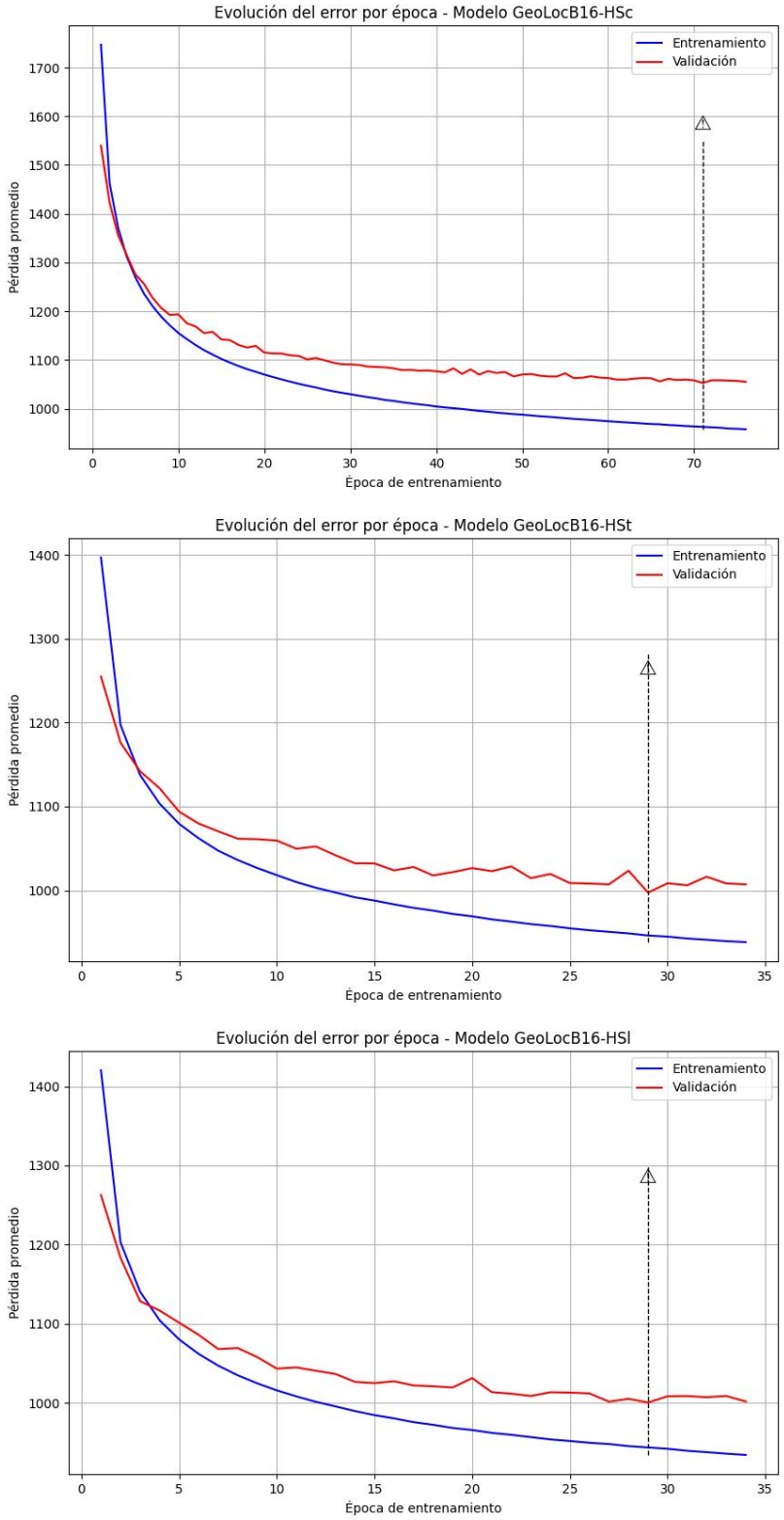

Figura A.8: Evolución de error - Modelos `GeoLocB16-HS`

A.1.9. B16-GeoscoreLoss

Hay tres modelos: `GeoLocB16-GSc`, `GeoLocB16-GSt` y `GeoLocB16-GSl`

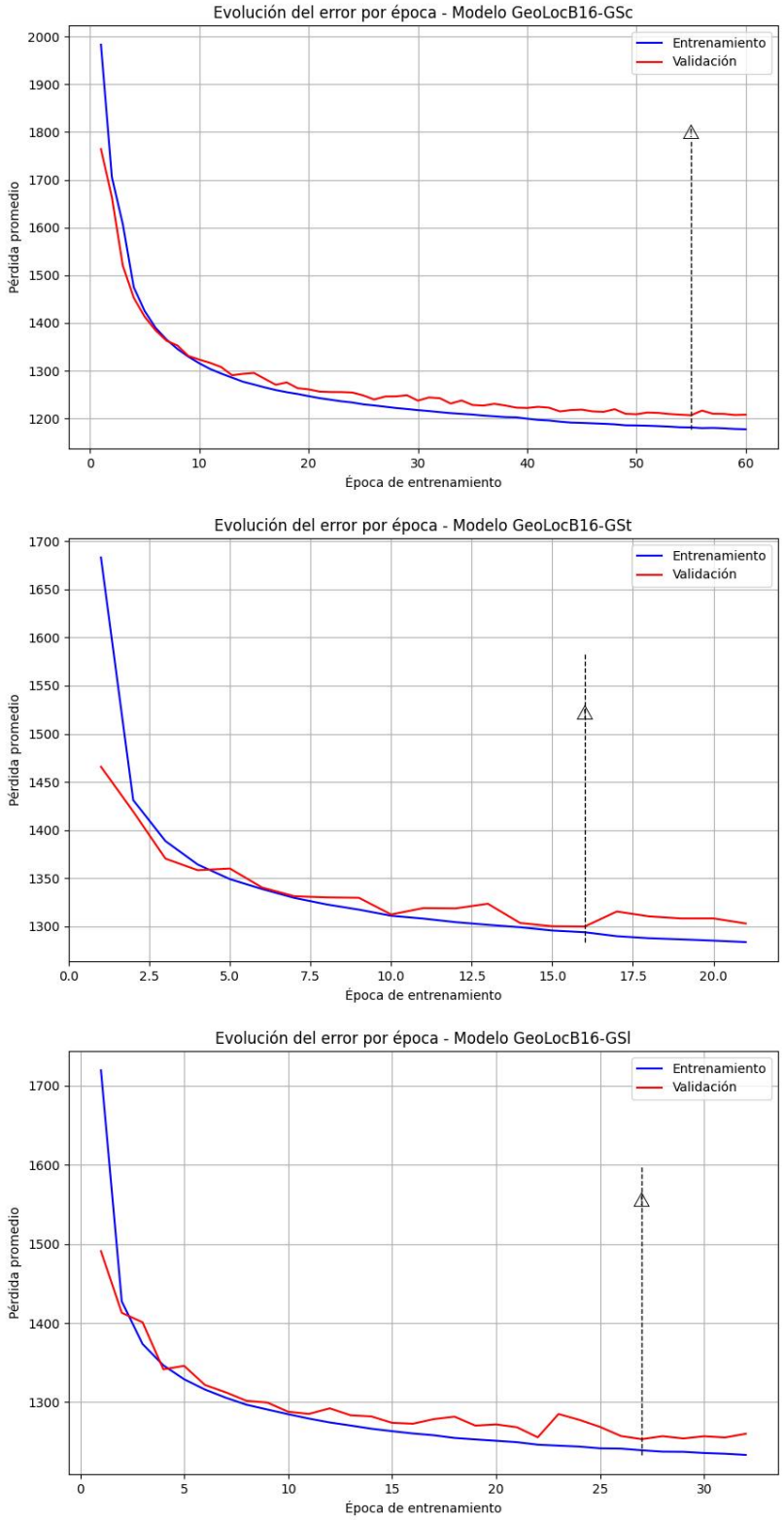

Figura A.9: Evolución de error - Modelos `GeoLocB16-GS`

A.1.10. B16-LinealScoreLoss

Hay tres modelos: `GeoLocB16-LSc`, `GeoLocB16-LSt` y `GeoLocB16-LSl`

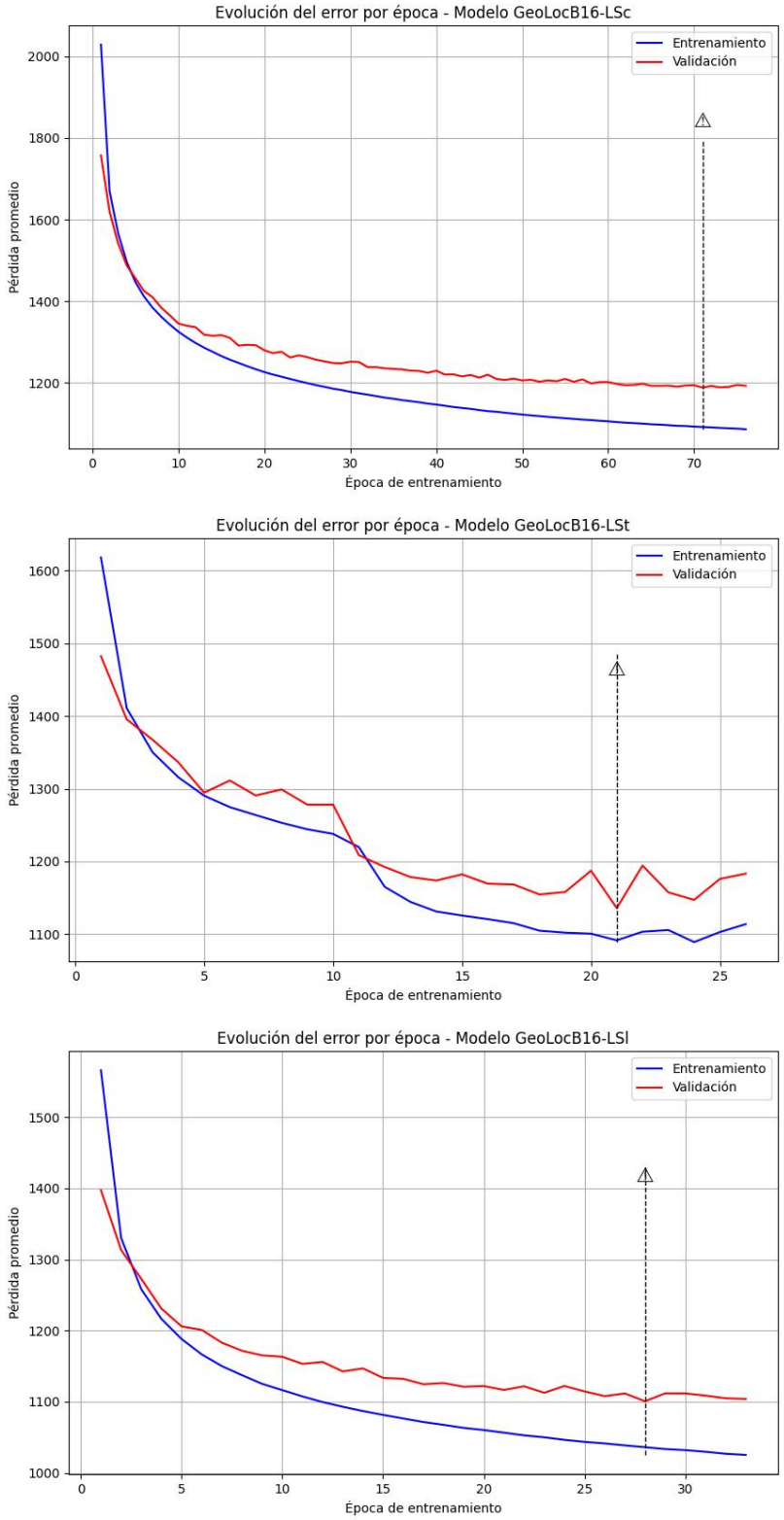

Figura A.10: Evolución de error - Modelos `GeoLocB16-LS`

A.1.11. B16-InterCubicMonotonusHermiteLoss

Hay tres modelos: `GeoLocB16-ICMHc`, `GeoLocB16-ICMHt` y `GeoLocB16-ICMHl`

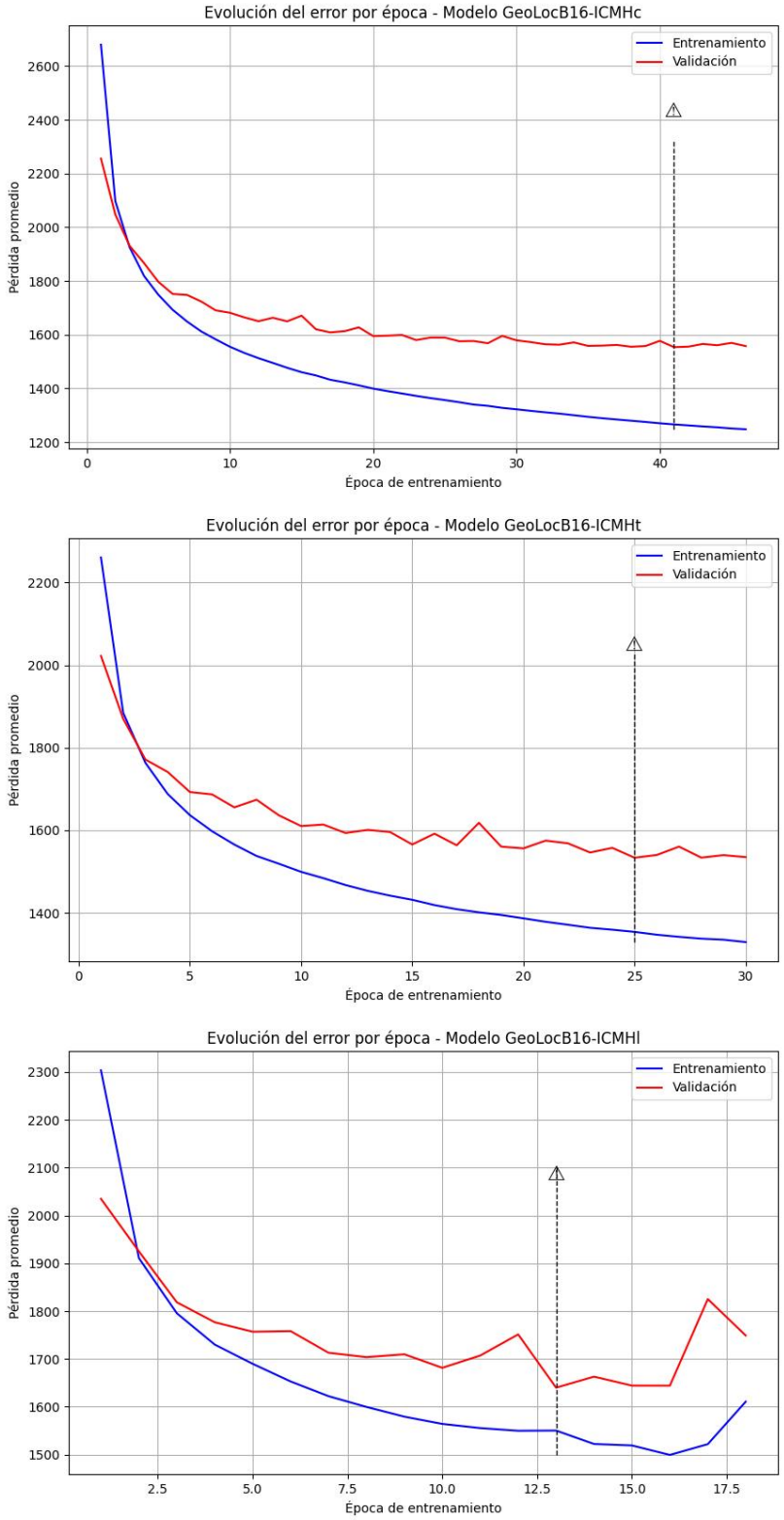

Figura A.11: Evolución de error - Modelos `GeoLocB16-ICMH`

A.1.12. B16-GeoscoreModificadoLoss

Hay tres modelos: `GeoLocB16-GSMc`, `GeoLocB16-GSMt` y `GeoLocB16-GSMl`

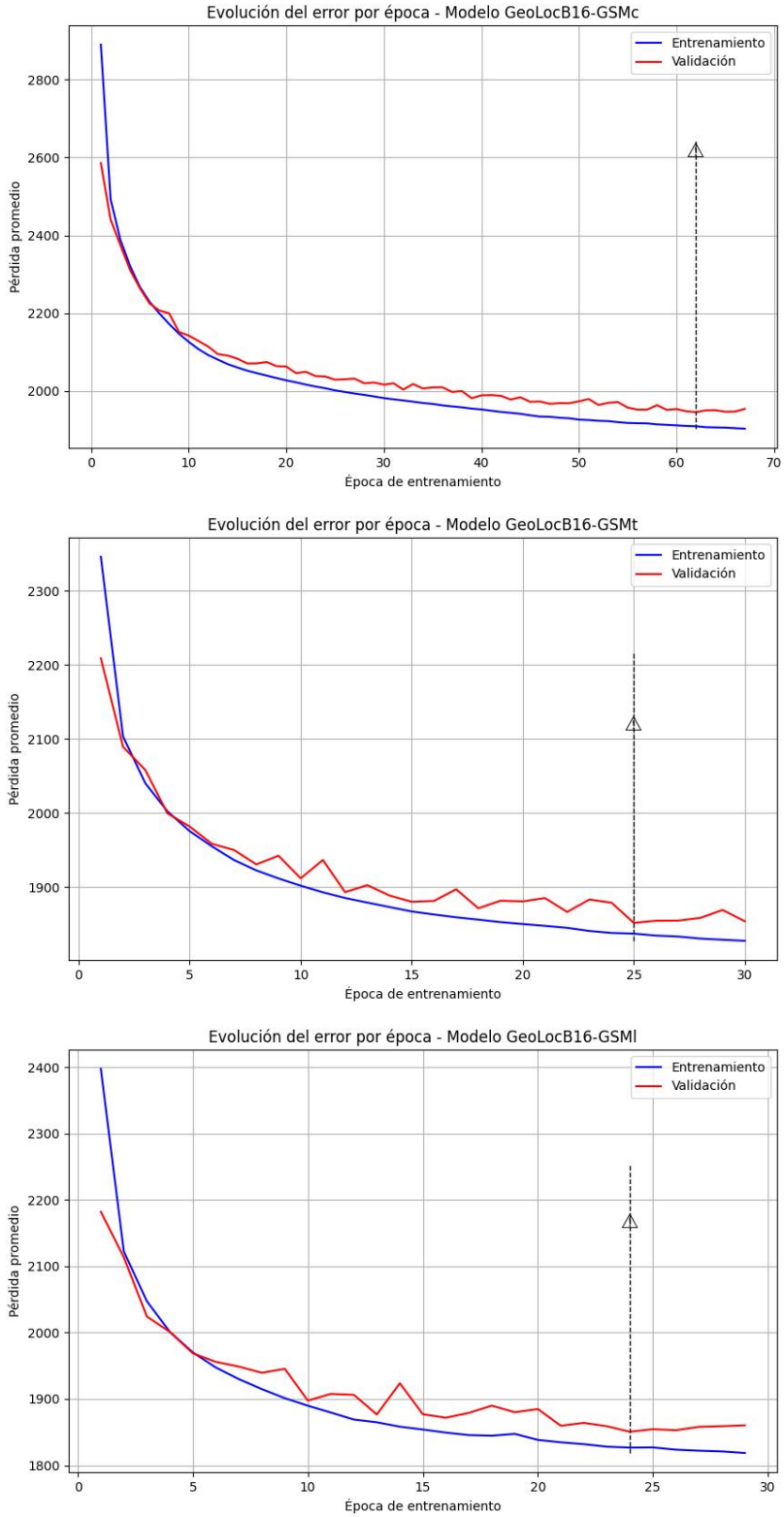

Figura A.12: Evolución de error - Modelos `GeoLocB16-GSM`

Apéndice B

Histogramas, CDFs y Resultados Adicionales

Este apéndice tiene como objetivo proporcionar material suplementario que documenta la evaluación de los diferentes modelos usando el conjunto de test y la distancia Haversine. Tal como se indicó en la Sección 4.5, el análisis del rendimiento equilibrado se realizará a partir de distintos tipos de gráficas: histogramas de errores geográficos y funciones de distribución acumulada (CDF) del error geográfico. Por ello, se incluyen en este apéndice todas las representaciones gráficas correspondientes, con el fin de exponer la información de manera estructurada y accesible. Finalmente, también se presentarán algunos resultados y comentarios adicionales observados durante la evaluación de los modelos.

B.1. Histogramas de Errores Geográficos

A continuación, se presentan los histogramas de errores geográficos correspondientes a todos los modelos, tanto aquellos que utilizan como *Image Encoder* la arquitectura `CLIP-ViT-B/32` como los de `CLIP-ViT-B/16`. Estas visualizaciones se organizarán a su vez en subsecciones diferenciadas según esa arquitectura utilizada y la métrica empleada durante el entrenamiento de cada modelo.

B.1.1. B32-MSELoss

Hay tres modelos: `GeoLocB32-MSEc`, `GeoLocB32-MSEt` y `GeoLocB32-MSEl`

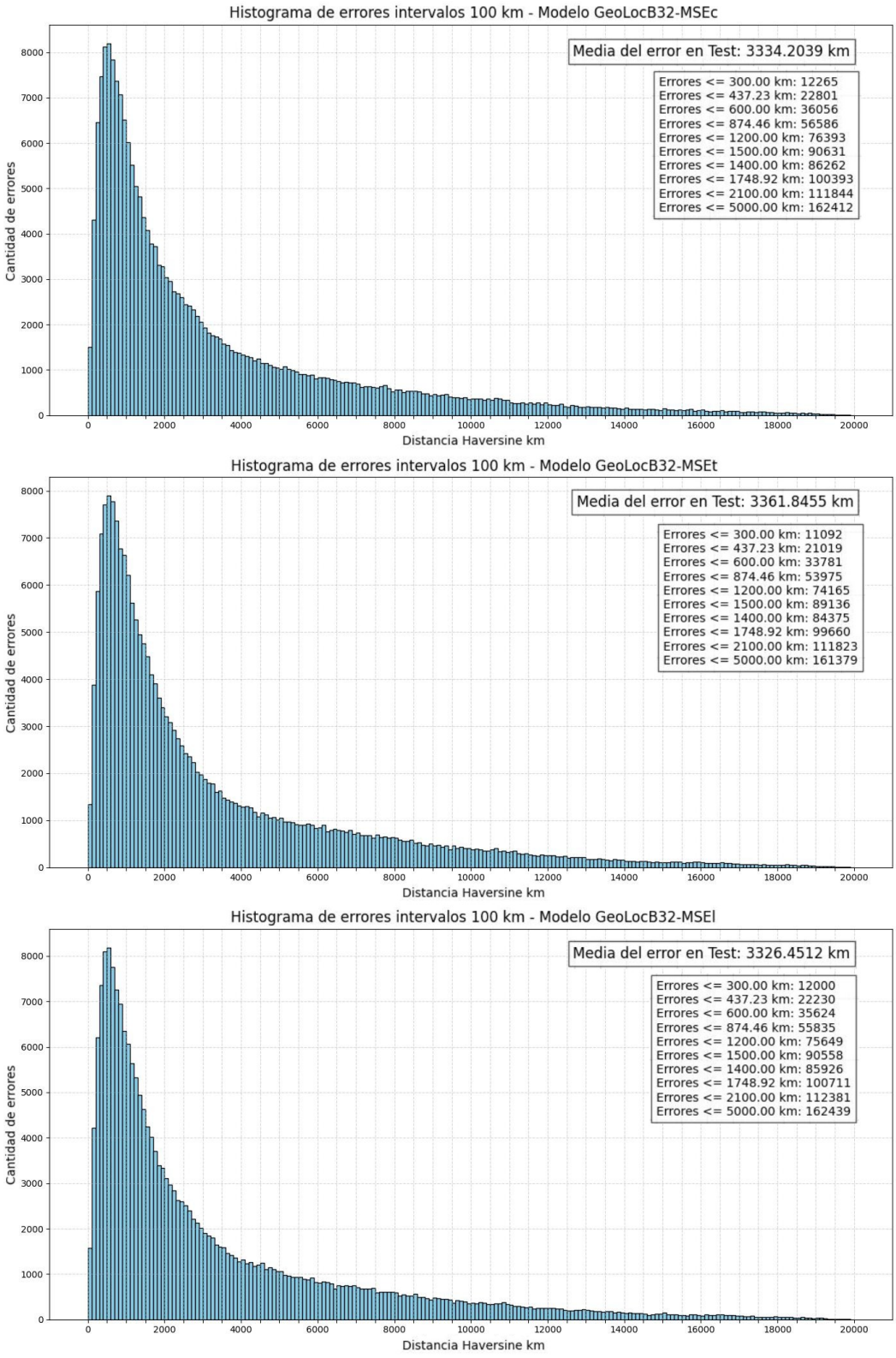

Figura B.1: Histogramas de errores intervalos 100 km - Modelos `GeoLoc32-MSE`

B.1.2. B32-HaversineLoss

Hay tres modelos: `GeoLocB32-HSc`, `GeoLocB32-HSt` y `GeoLocB32-HSl`

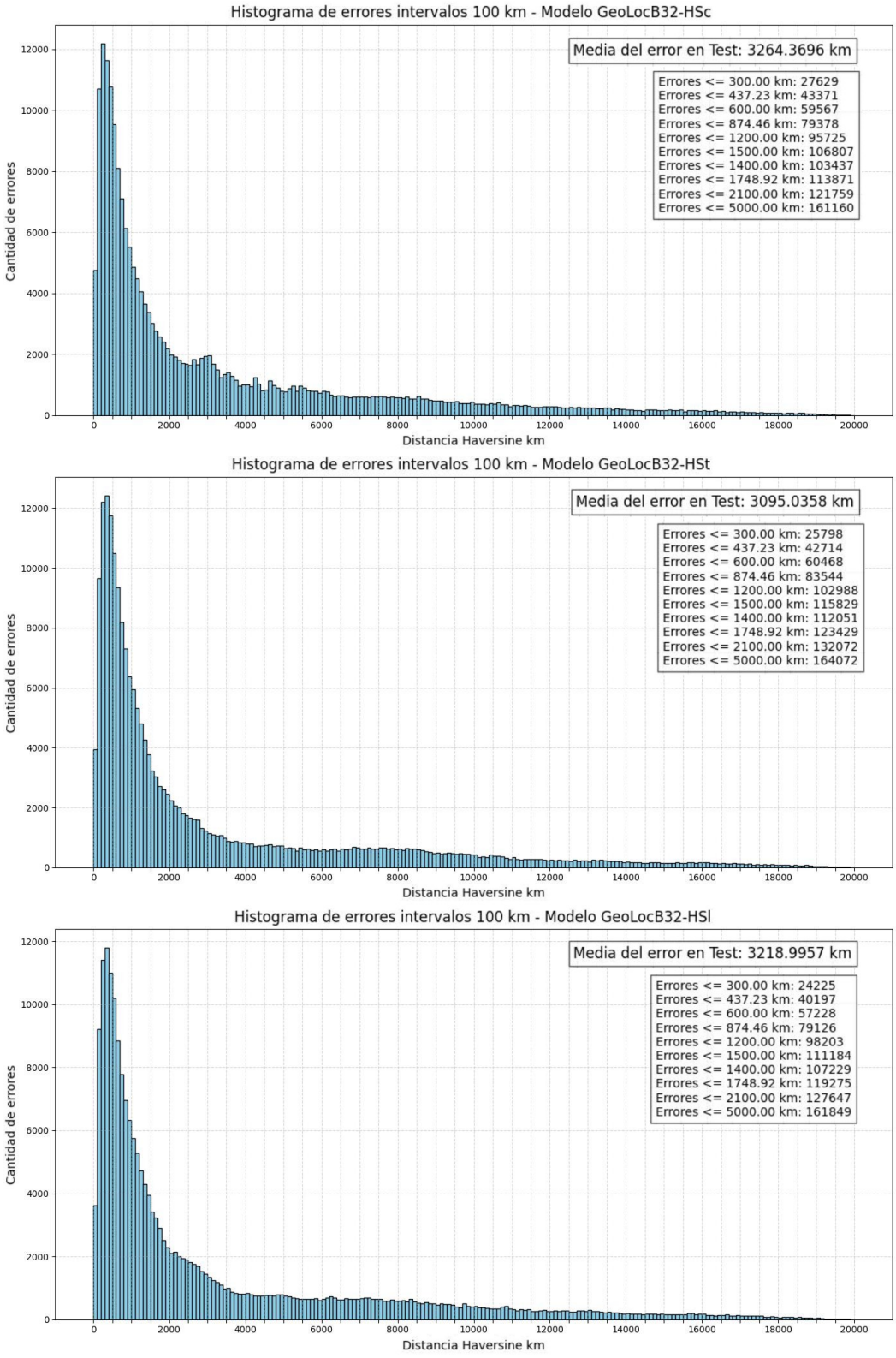

Figura B.2: Histogramas de errores intervalos 100 km - Modelos `GeoLocB32-HS`

B.1.3. B32-GeoscoreLoss

Hay tres modelos: `GeoLocB32-GSc`, `GeoLocB32-GSt` y `GeoLocB32-GSl`

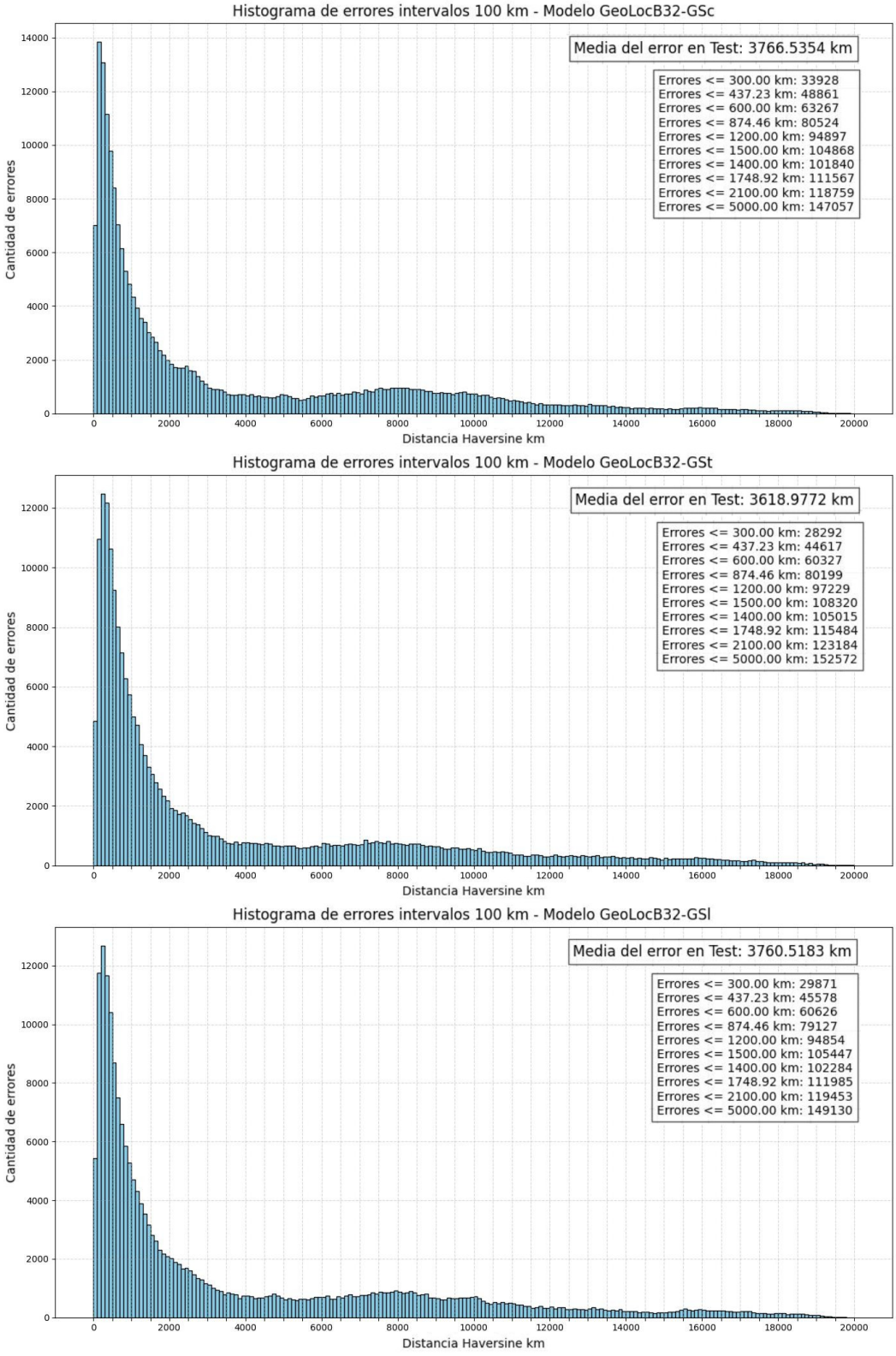

Figura B.3: Histogramas de errores intervalos 100 km - Modelos `GeoLocB32-GS`

B.1.4. B32-LinealScoreLoss

Hay tres modelos: `GeoLocB32-LSc`, `GeoLocB32-LSt` y `GeoLocB32-LSl`

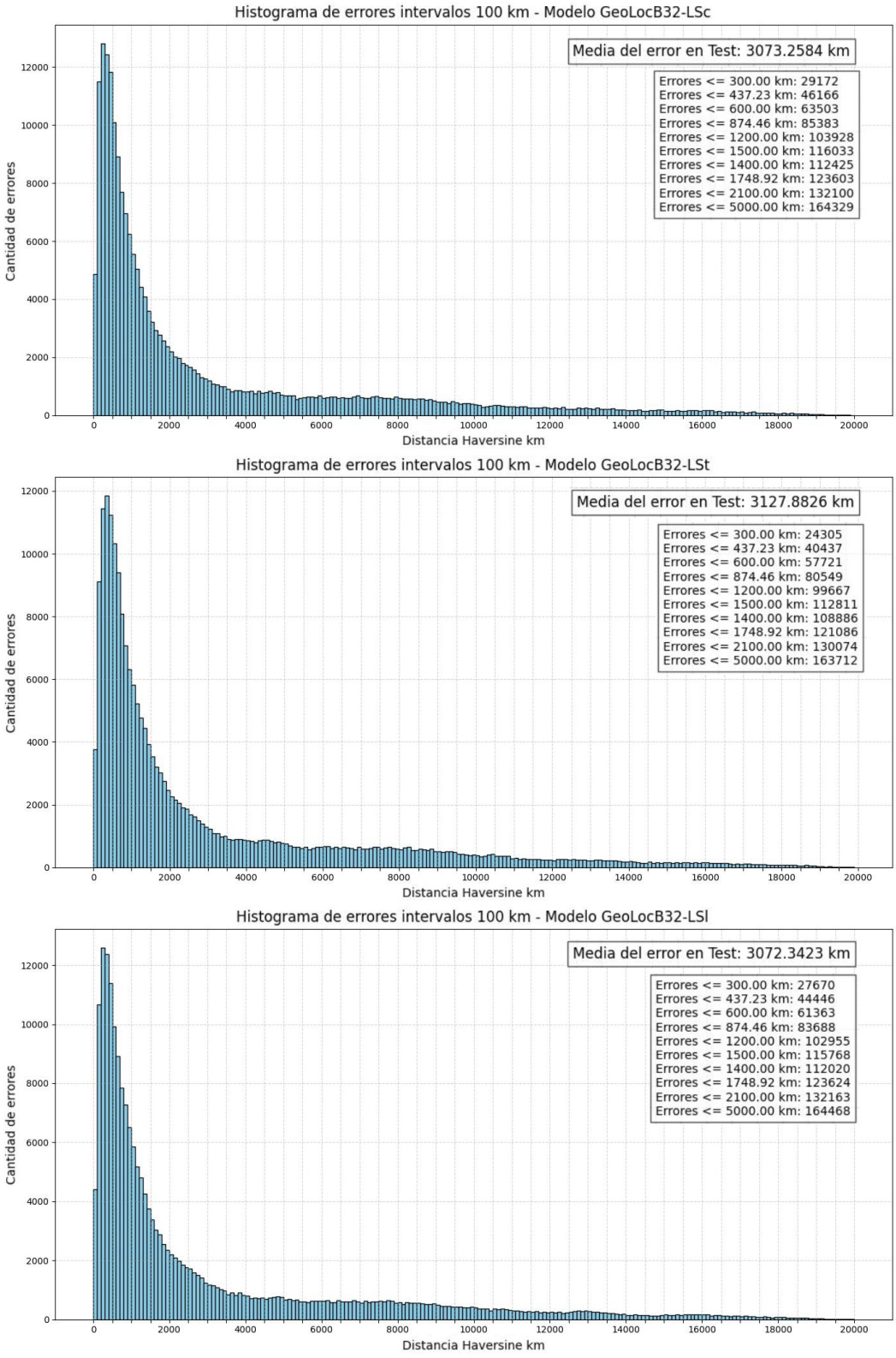

Figura B.4: Histogramas de errores intervalos 100 km - Modelos `GeoLocB32-LS`

B.1.5. B32-InterCubicMonotonusHermiteLoss

Hay tres modelos: `GeoLocB32-ICMHc`, `GeoLocB32-ICMHt` y `GeoLocB32-ICMHl`

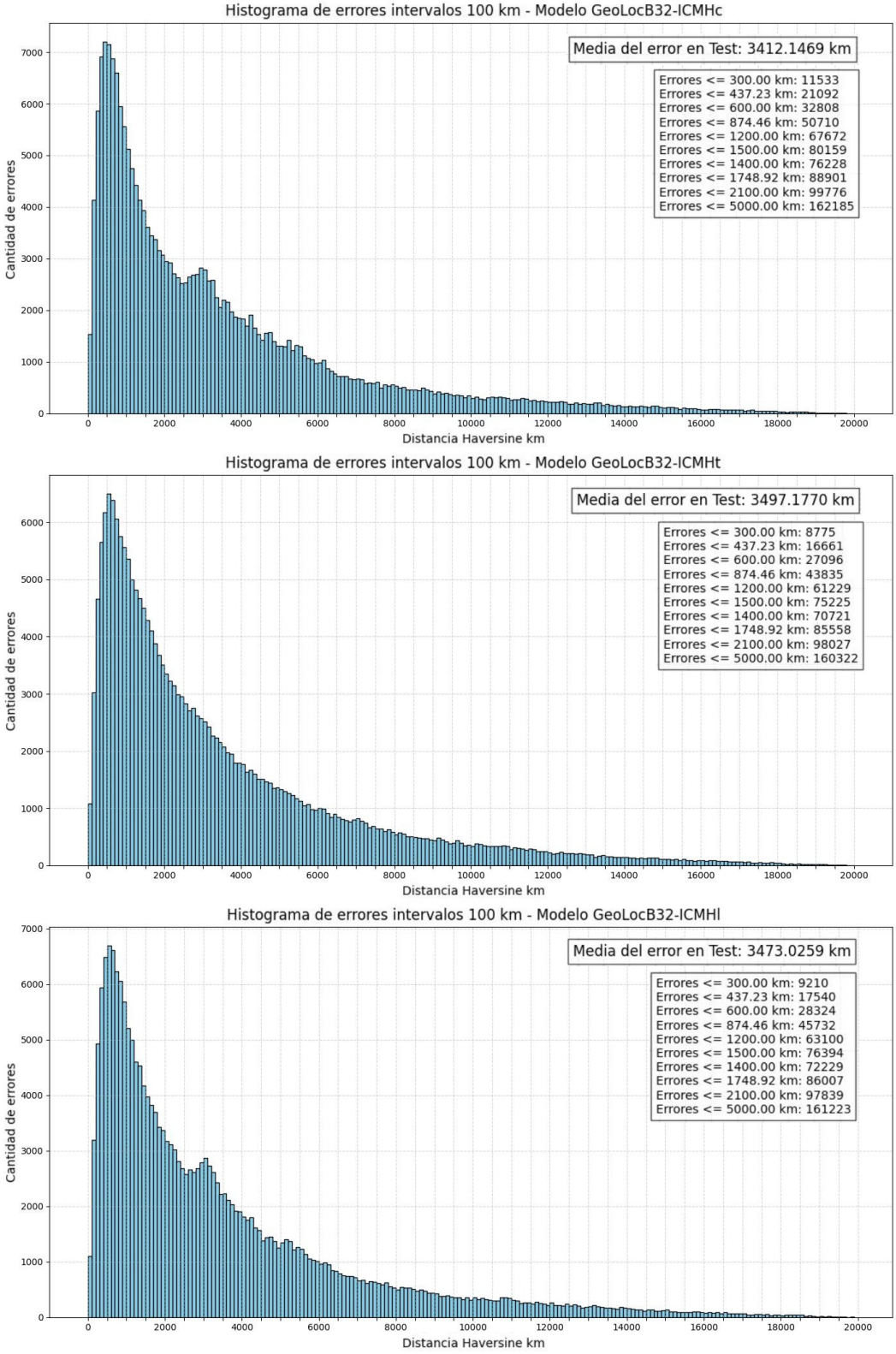

Figura B.5: Histogramas de errores intervalos 100 km - Modelos `GeoLocB32-ICMH`

B.1.6. B32-GeoscoreModificadoLoss

Hay tres modelos: `GeoLocB32-GSMc`, `GeoLocB32-GSMt` y `GeoLocB32-GSMl`

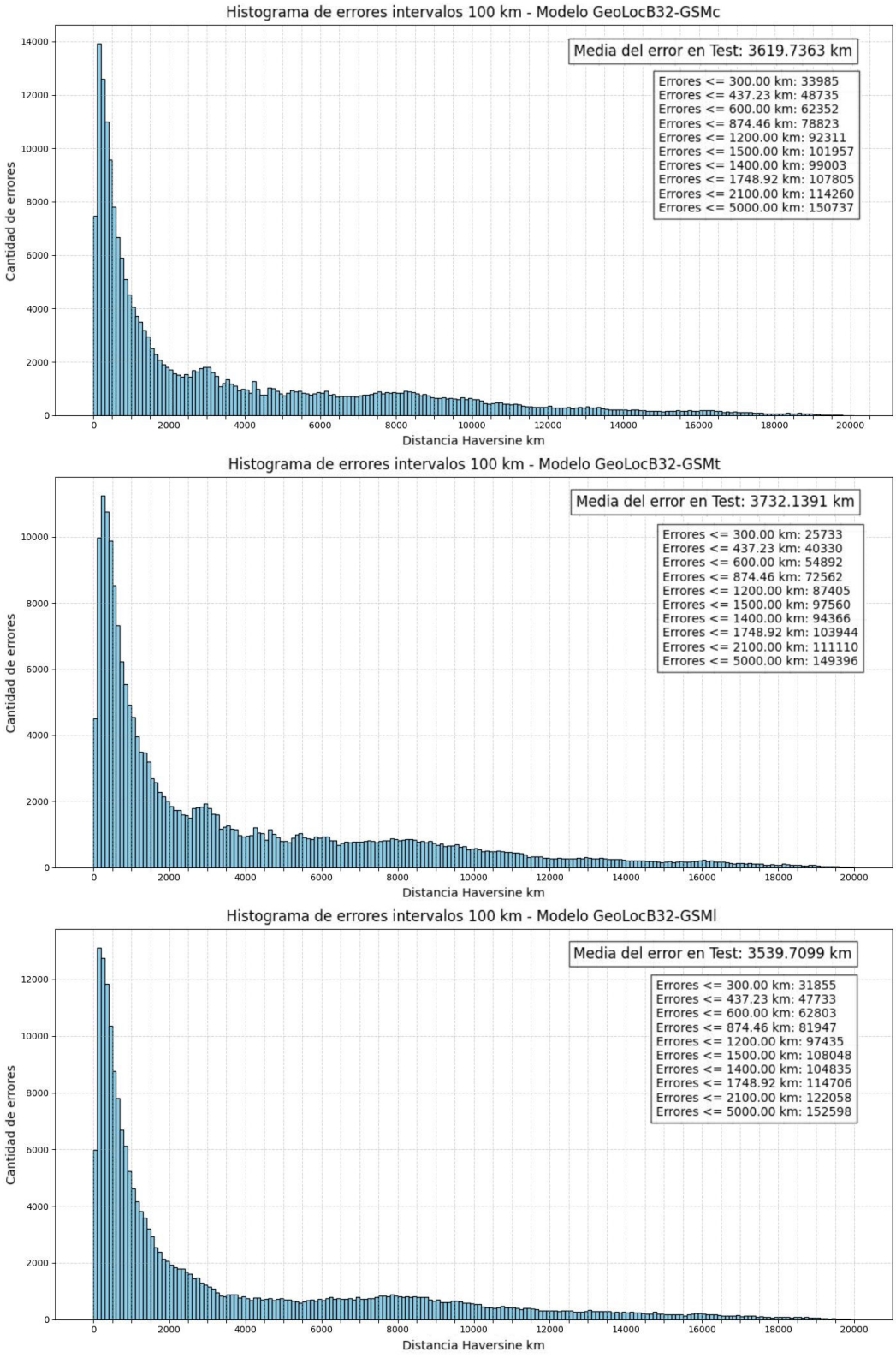

Figura B.6: Histogramas de errores intervalos 100 km - Modelos `GeoLocB32-GSM`

B.1.7. B16-MSELoss

Hay tres modelos: `GeoLocB16-MSEc`, `GeoLocB16-MSEt` y `GeoLocB16-MSEl`

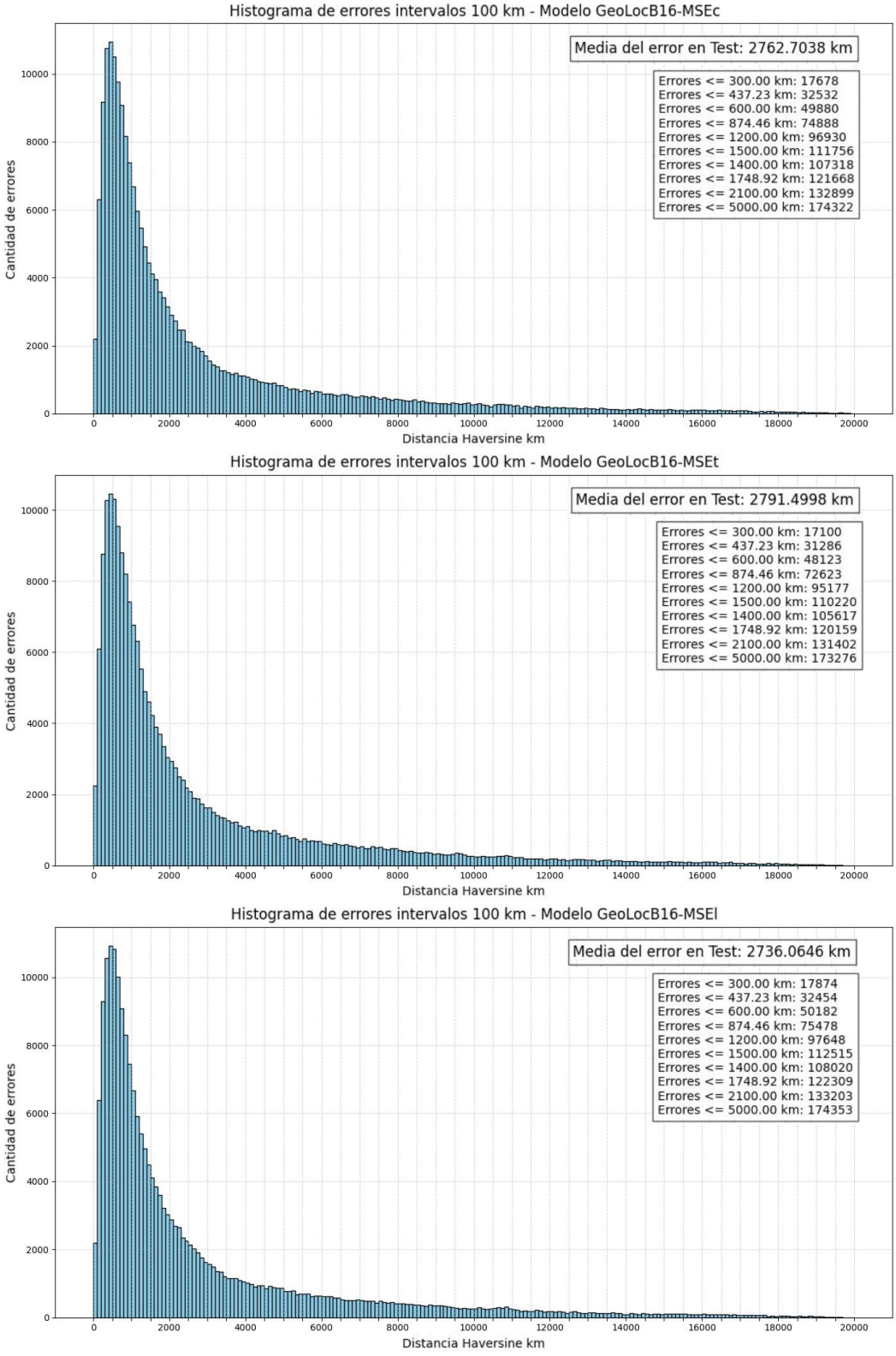

Figura B.7: Histogramas de errores intervalos 100 km - Modelos `GeoLocB16-MSE`

B.1.8. B16-HaversineLoss

Hay tres modelos: `GeoLocB16-HSc`, `GeoLocB16-HSt` y `GeoLocB16-HSl`

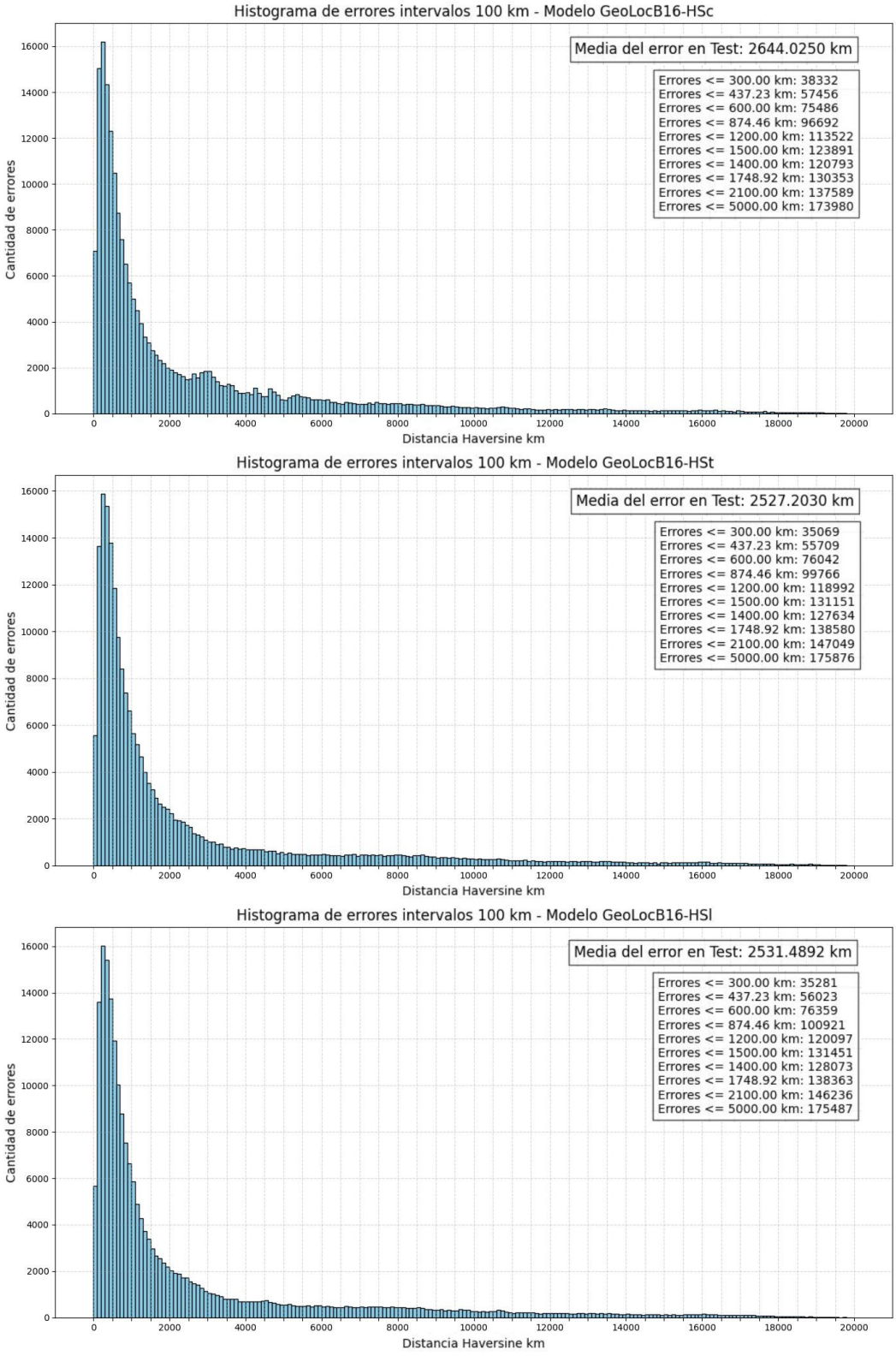

Figura B.8: Histogramas de errores intervalos 100 km - Modelos `GeoLocB16-HS`

B.1.9. B16-GeoscoreLoss

Hay tres modelos: `GeoLocB16-GSc`, `GeoLocB16-GSt` y `GeoLocB16-GSl`

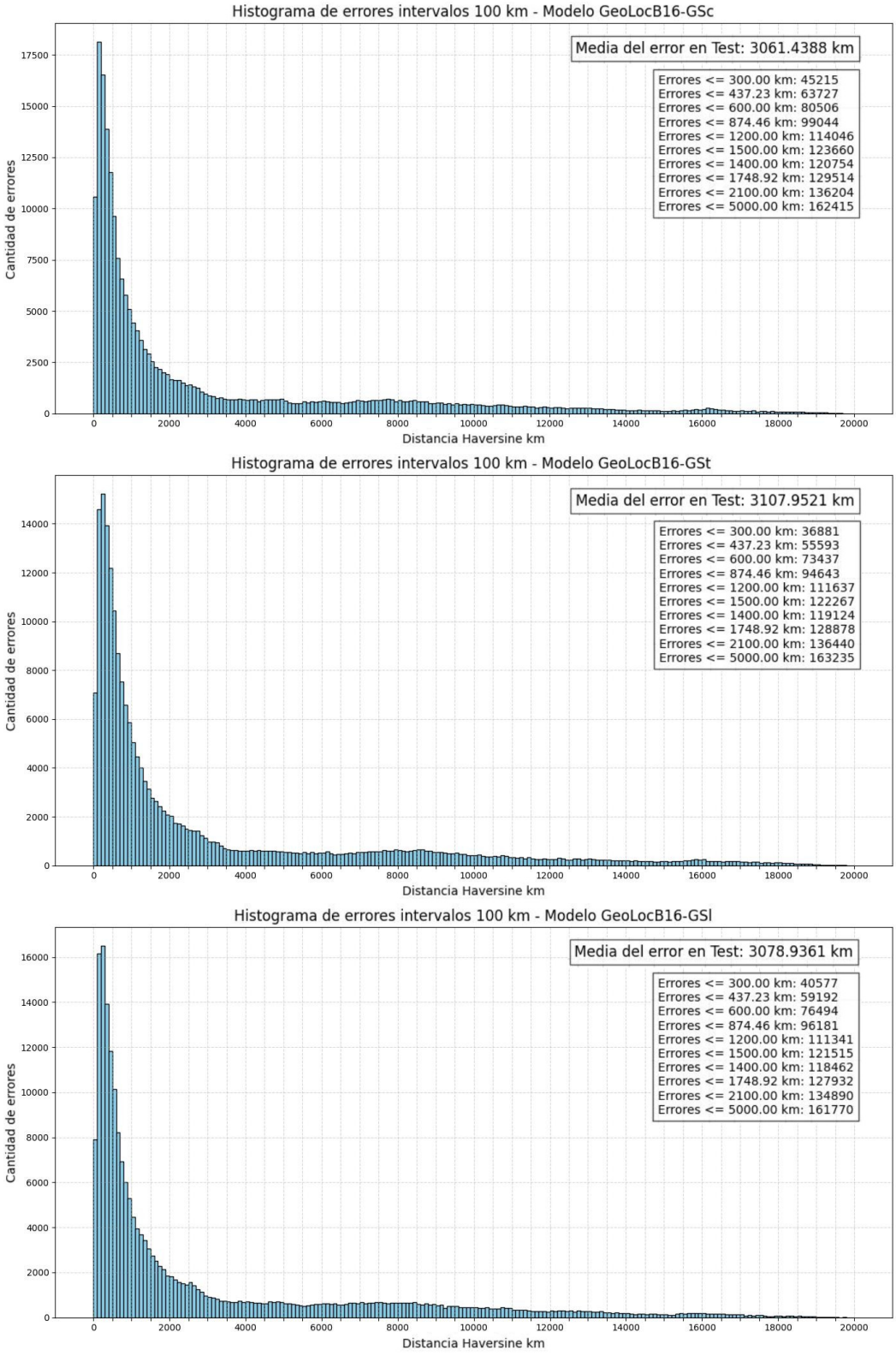

Figura B.9: Histogramas de errores intervalos 100 km - Modelos `GeoLocB16-GS`

B.1.10. B16-LinealScoreLoss

Hay tres modelos: `GeoLocB16-LSc`, `GeoLocB16-LSt` y `GeoLocB16-LSl`

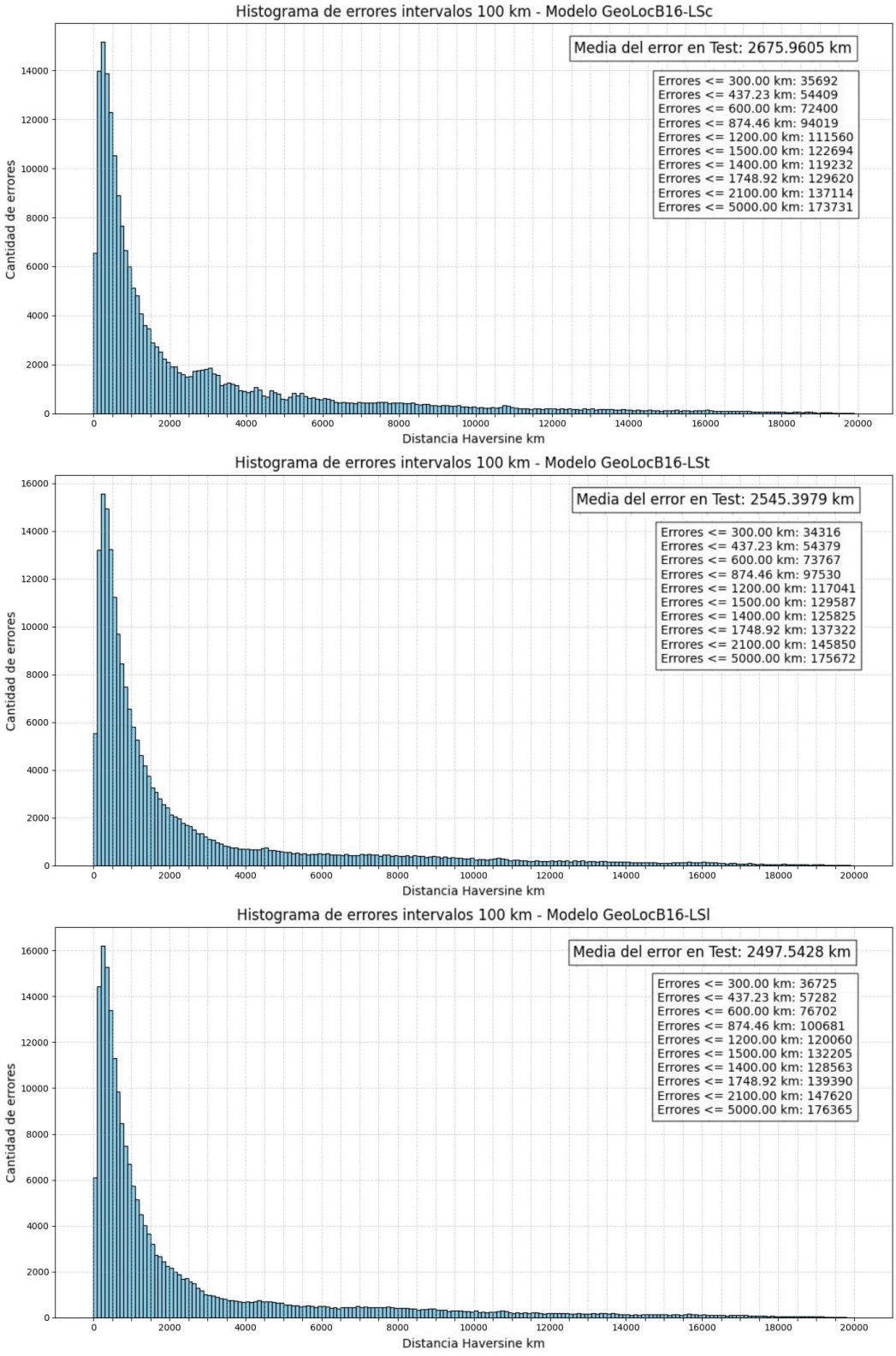

Figura B.10: Histogramas de errores intervalos 100 km - Modelos `GeoLocB16-LS`

B.1.11. B16-InterCubicMonotonusHermiteLoss

Hay tres modelos: `GeoLocB16-ICMHc`, `GeoLocB16-ICMHt` y `GeoLocB16-ICMHl`

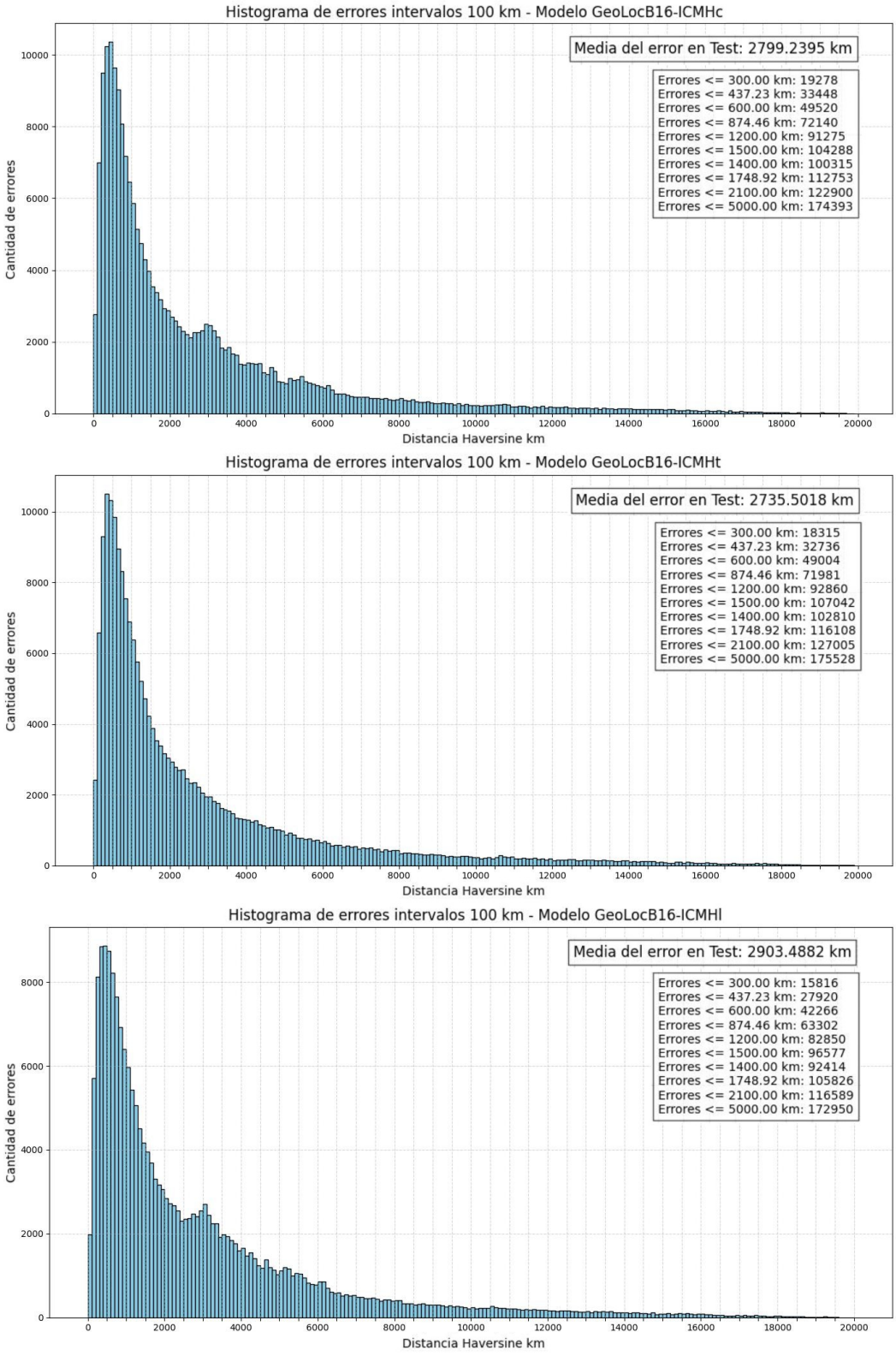

Figura B.11: Histogramas de errores intervalos 100 km - Modelos `GeoLocB16-ICMH`

B.1.12. B16-GeoscoreModificadoLoss

Hay tres modelos: `GeoLocB16-GSMc`, `GeoLocB16-GSMt` y `GeoLocB16-GSMl`

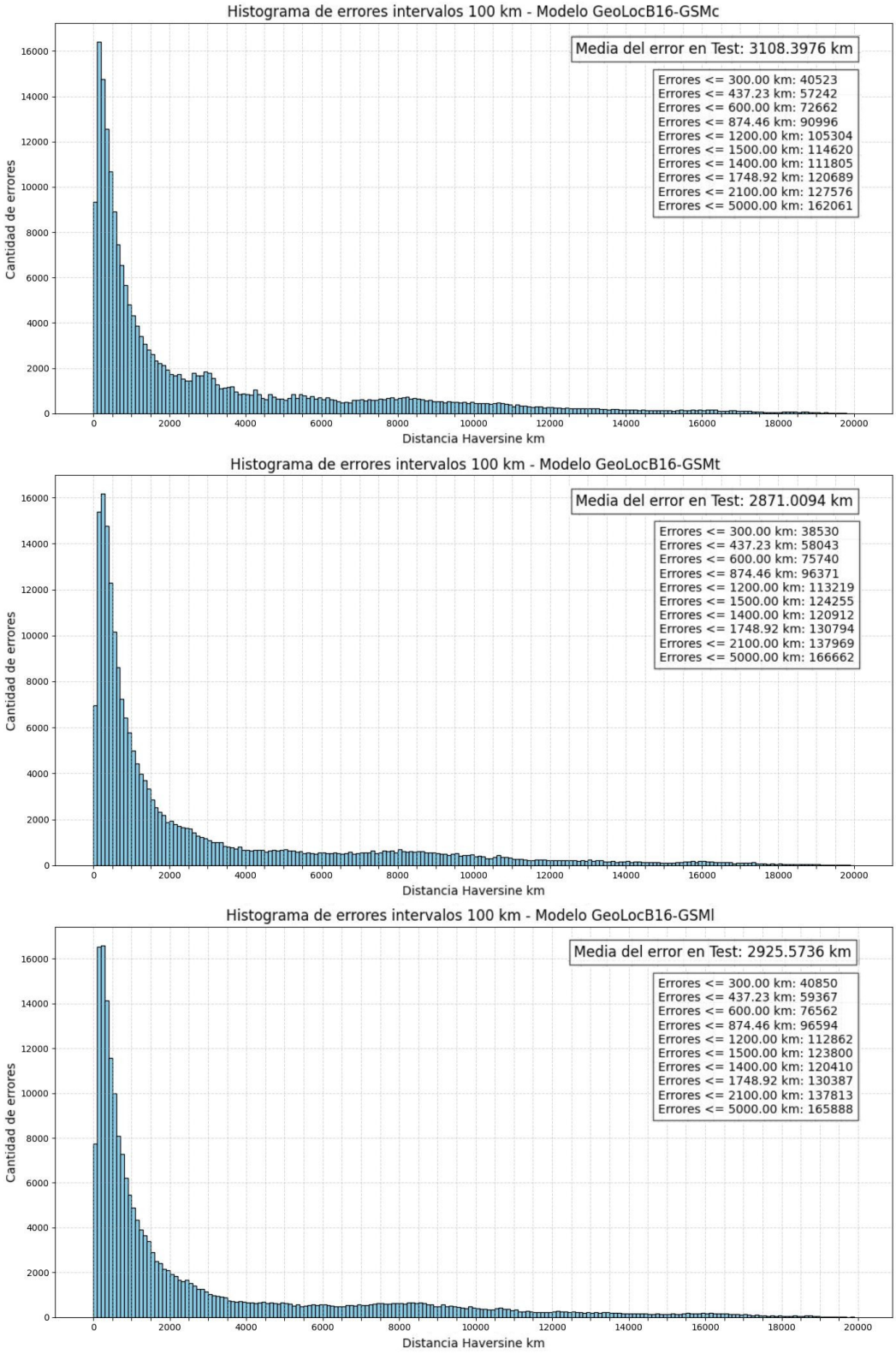

Figura B.12: Histogramas de errores intervalos 100 km - Modelos `GeoLocB16-GSM`

B.2. Funciones de Distribución Acumulada (CDF)

A continuación, se presentan las versiones ampliadas de las gráficas de función de distribución acumulada mostradas en la Sección 4.5.2.

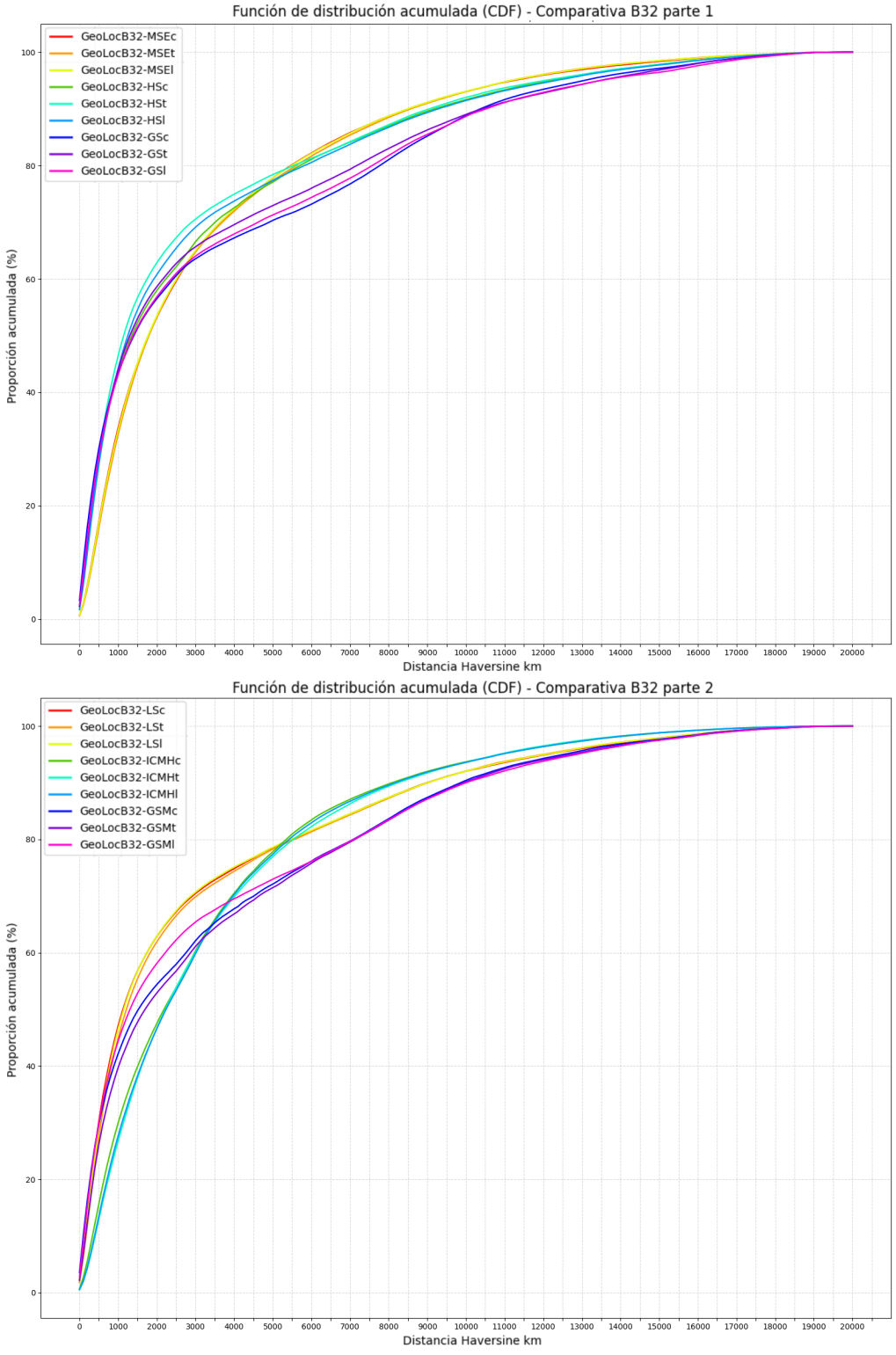

Figura B.13: Funciones de distribución acumulada (CDF) de todos los modelos B32

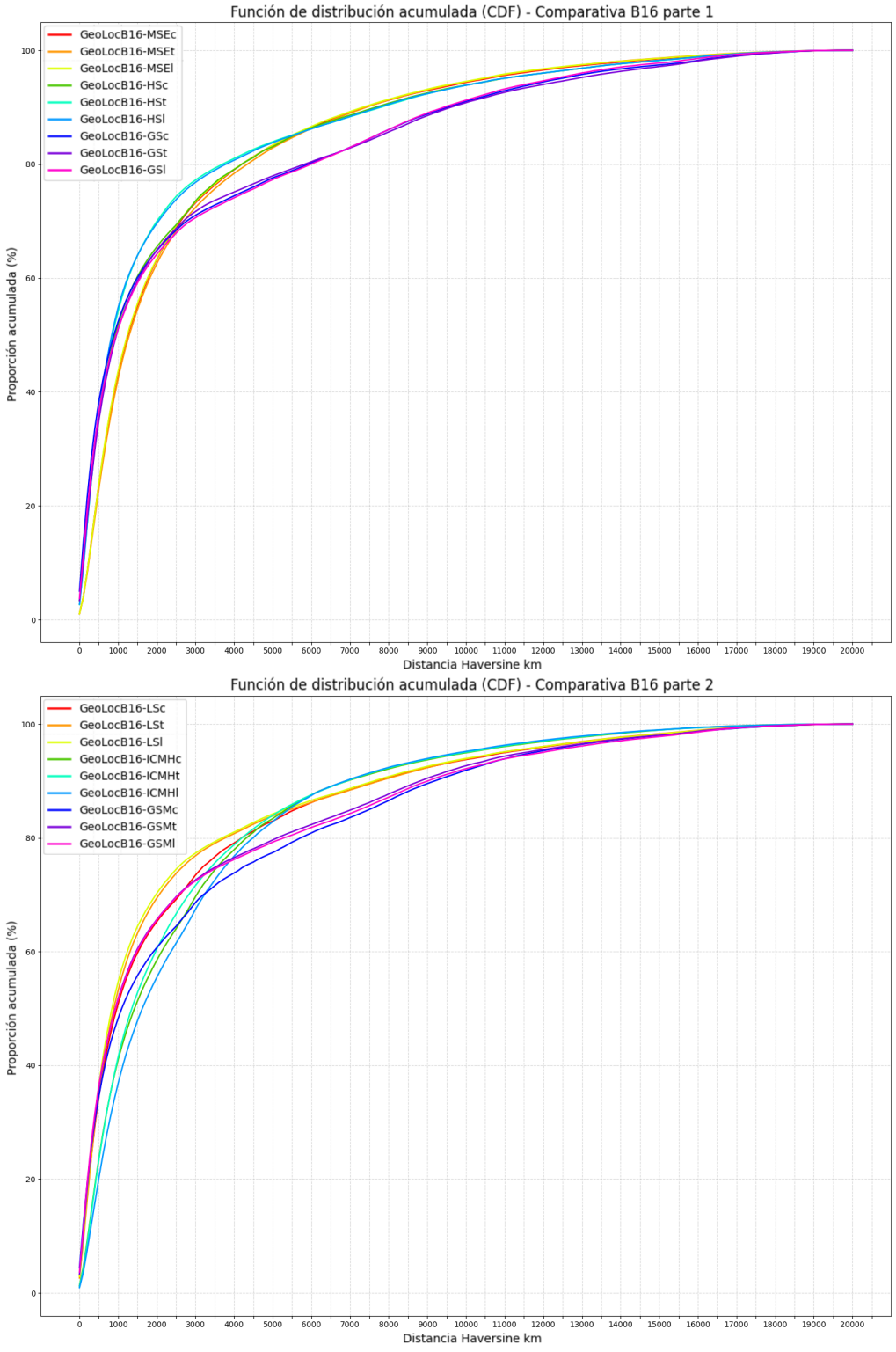

Figura B.14: Funciones de distribución acumulada (CDF) de todos los modelos B16

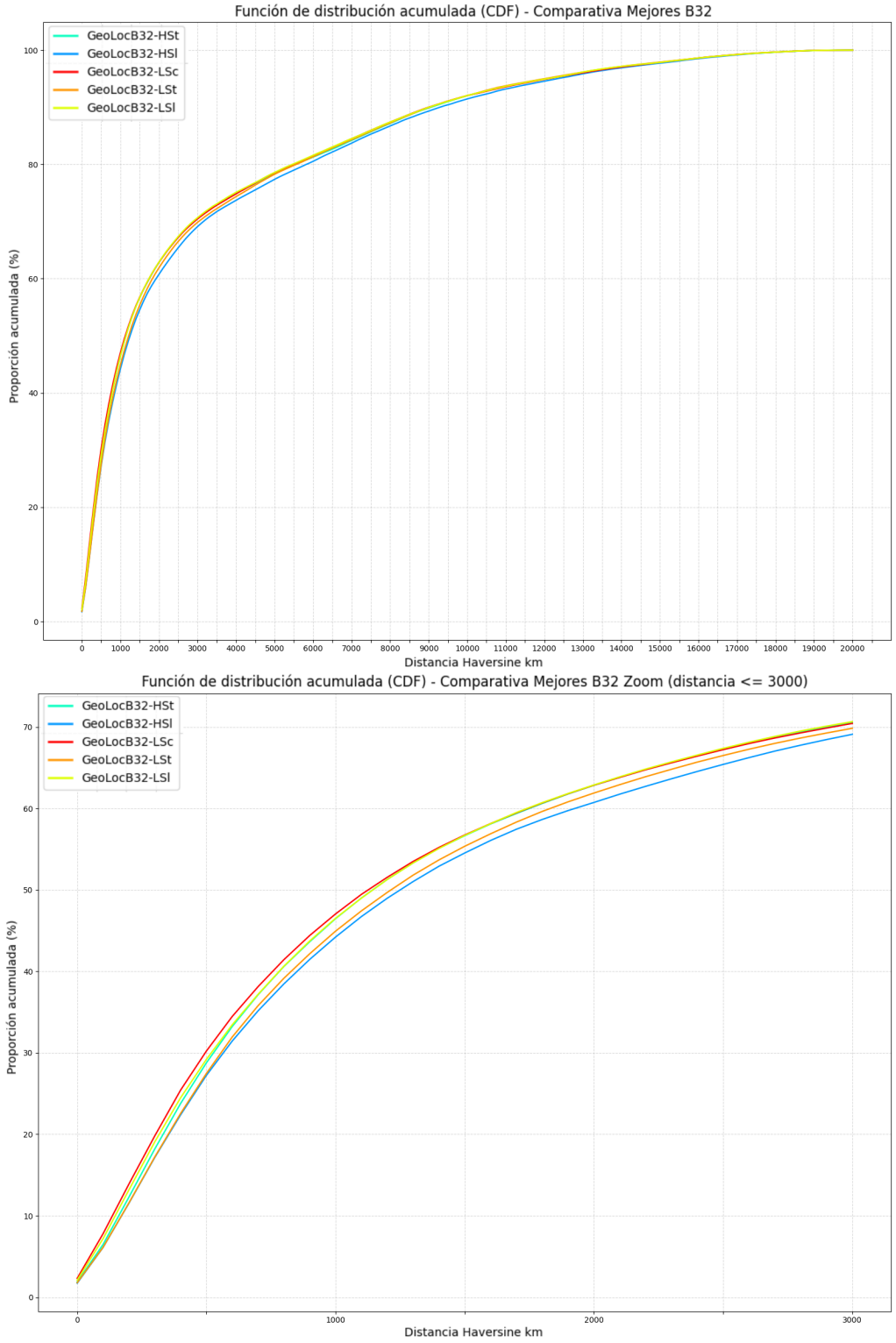

Figura B.15: Mejores funciones de distribución acumulada de los modelos basados en `ViT-B/32`

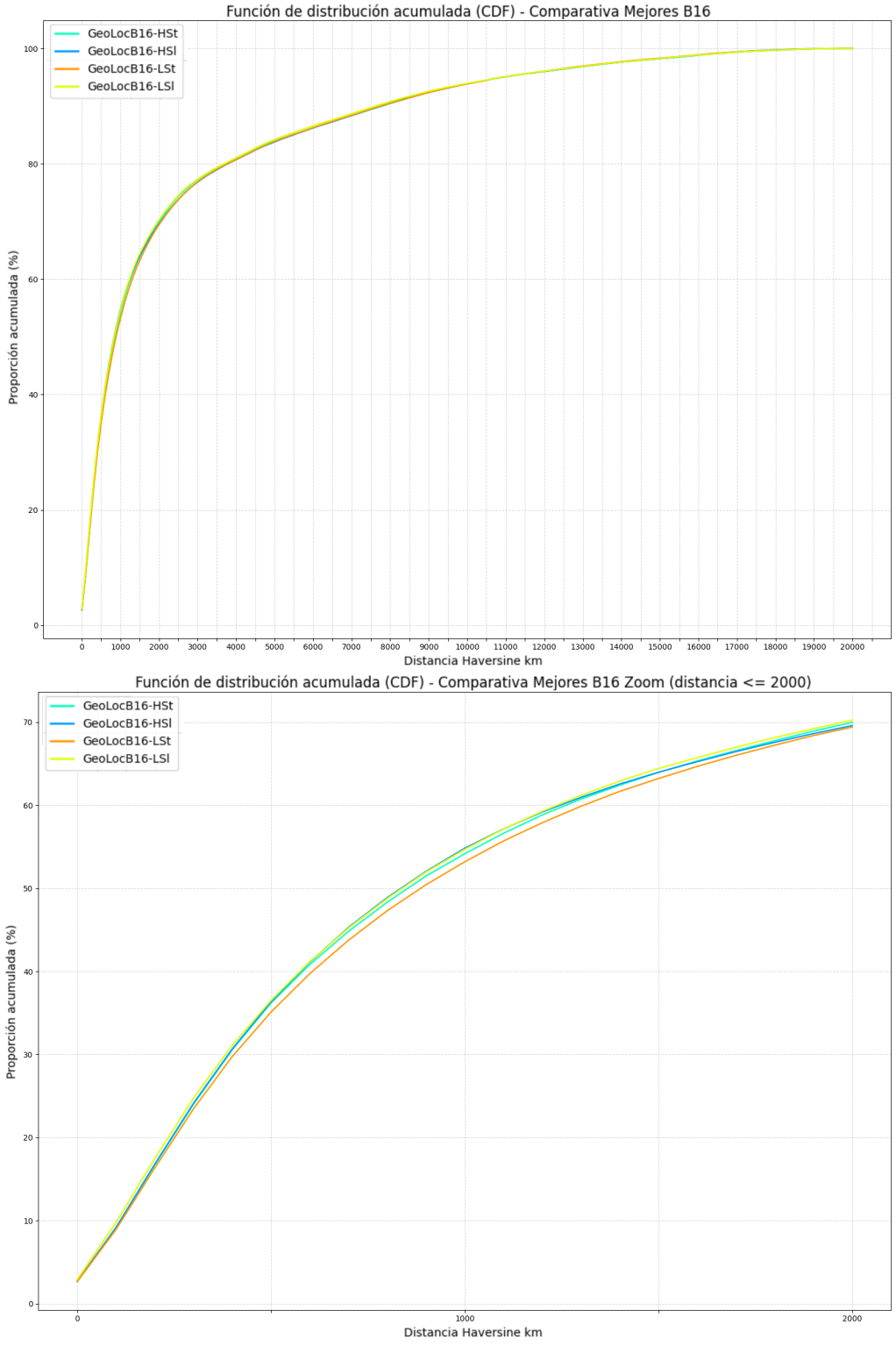

Figura B.16: Mejores funciones de distribución acumulada de los modelos basados en `ViT-B/16`

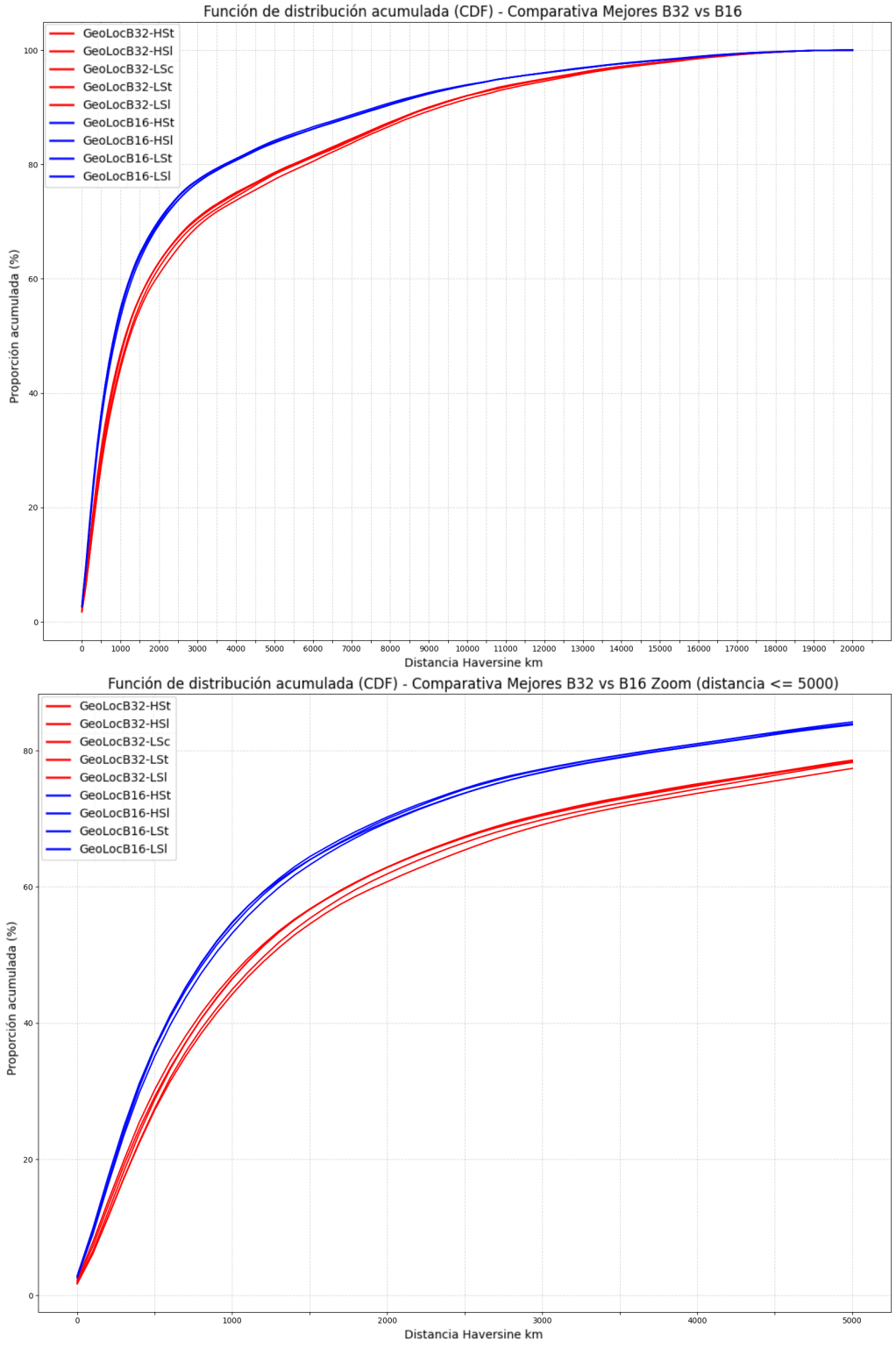

Figura B.17: Comparación mejores funciones de distribución acumulada de los modelos B32 vs B16

B.3. Resultados Adicionales

En esta sección se analizarán en más detalle los modelos que, en una fase inicial, consiguen acertar más imágenes que el resto. Como ya se anticipó en la Sección 4.5.3, estos modelos van a ser los que fueron entrenados con las funciones de pérdida `GeoscoreLoss` (GS) y `GeoscoreModificadoLoss` (GSM).

Estas métricas incorporan la estrategia particular de penalizar con mayor intensidad los errores cuando las predicciones están cerca del objetivo y reducir progresivamente esa penalización a medida que la distancia aumenta. Bajo esta lógica, se prioriza estar "muy cerca" del lugar correcto frente a simplemente estar "menos lejos". Esta filosofía consigue en el entrenamiento afinar más las predicciones cercanas, permitiendo a los modelos obtener una mayor proporción acumulada de imágenes en los tramos iniciales.

Para verificar esta hipótesis, nos vamos a fijar en las funciones de distribución acumulada (CDF) centradas en las primeras distancias, como se muestra en las Figuras B.19 y B.20. Estas gráficas presentan un *zoom* a distancias ≤ 1000 km para todos los modelos con arquitecturas `CLIP-ViT-B/32` y `CLIP-ViT-B/16` respectivamente.

A partir de estas figuras, observando los crecimientos iniciales más rápidos y pronunciados, podemos identificar dentro de cada tipo de arquitectura los modelos que consiguen acertar en esa fase inicial más imágenes que el resto. Estos se recogen en la Figura B.21 para los modelos `ViT-B/32`, y en la Figura B.22 para los modelos `ViT-B/16`. En ambas imágenes, la gráfica de arriba muestra la CDF completa, mientras que la de abajo presenta un *zoom* en los primeros tramos.

Con apoyo en esas gráficas de *zoom*, se va a elegir el mejor modelo de cada grupo. El criterio seguido es seleccionar aquel modelo que, desde el comienzo y durante un tramo significativo inicial, presente la CDF más elevada, es decir, el que consigue clasificar correctamente el mayor número de imágenes a corta distancia. Aplicando este criterio, se identifican como mejores modelos a `GeoLocB32-GSMc` (para `ViT-B/32`) y a `GeoLocB16-GSc` (para `ViT-B/16`).

Finalmente, nos quedamos con `GeoLocB16-GSc`, ya que se puede ver en la Figura B.23 una clara superioridad de los modelos basados en `CLIP-ViT-B/16` (en azul) frente a los que usan el `CLIP-ViT-B/32` (en rojo).

El modelo `GeoLocB16-GSc` resulta ser el que obtiene una mayor proporción acumulada de imágenes en las distancias iniciales. Según el histograma de la Figura B.18, logra clasificar correctamente cerca del 25 % del conjunto de test (45 215 de 210 122 imágenes) a una distancia menor de 300 km. Esta cifra supera ampliamente a otros modelos entrenados con otras funciones, que apenas alcanzan las 38 520 imágenes en ese mismo umbral.

Además, si nos centramos en las dos primeras barras del histograma, se confirma su destacada precisión en los tramos iniciales: más de 10 000 imágenes a menos de 100 km y más de 17 500 entre $100 - 200$ km. Ningún otro modelo con funciones de pérdida distintas a GS o GSM consigue superar los 7500 y 15 000 respectivamente en esos mismos intervalos.

Todos estos excelentes resultados en las primeras distancias mencionados se pueden ver comparando la Figura B.18 con los histogramas de la Sección B.1.

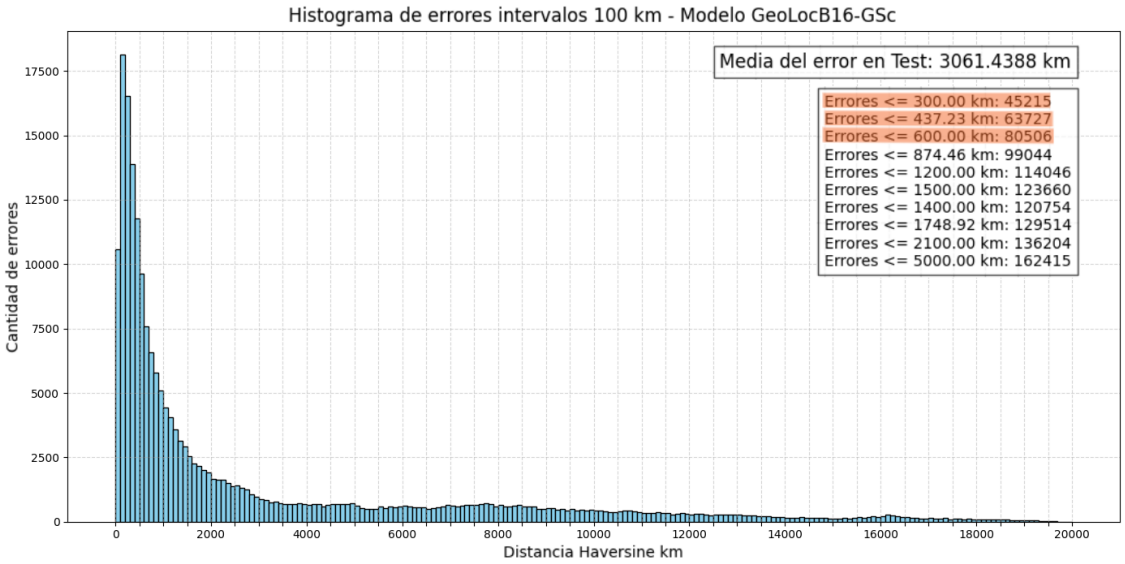

Figura B.18: Histograma del modelo GeoLoc

Sin embargo, esta ventaja inicial tiene una contrapartida. Como se anticipó anteriormente en la Sección 4.5.3 y se confirma en las Figuras B.19 y B.20, los modelos entrenados con funciones `Geoscore` presentan un rendimiento decreciente a medida que las distancias aumentan. Aunque lideran en los primeros tramos, terminan siendo superados por otros modelos, como los entrenados con las métricas `HaversineLoss` y `LinealScoreLoss`, que penalizan de forma más uniforme y logran predicciones más competitivas en rangos más amplios.

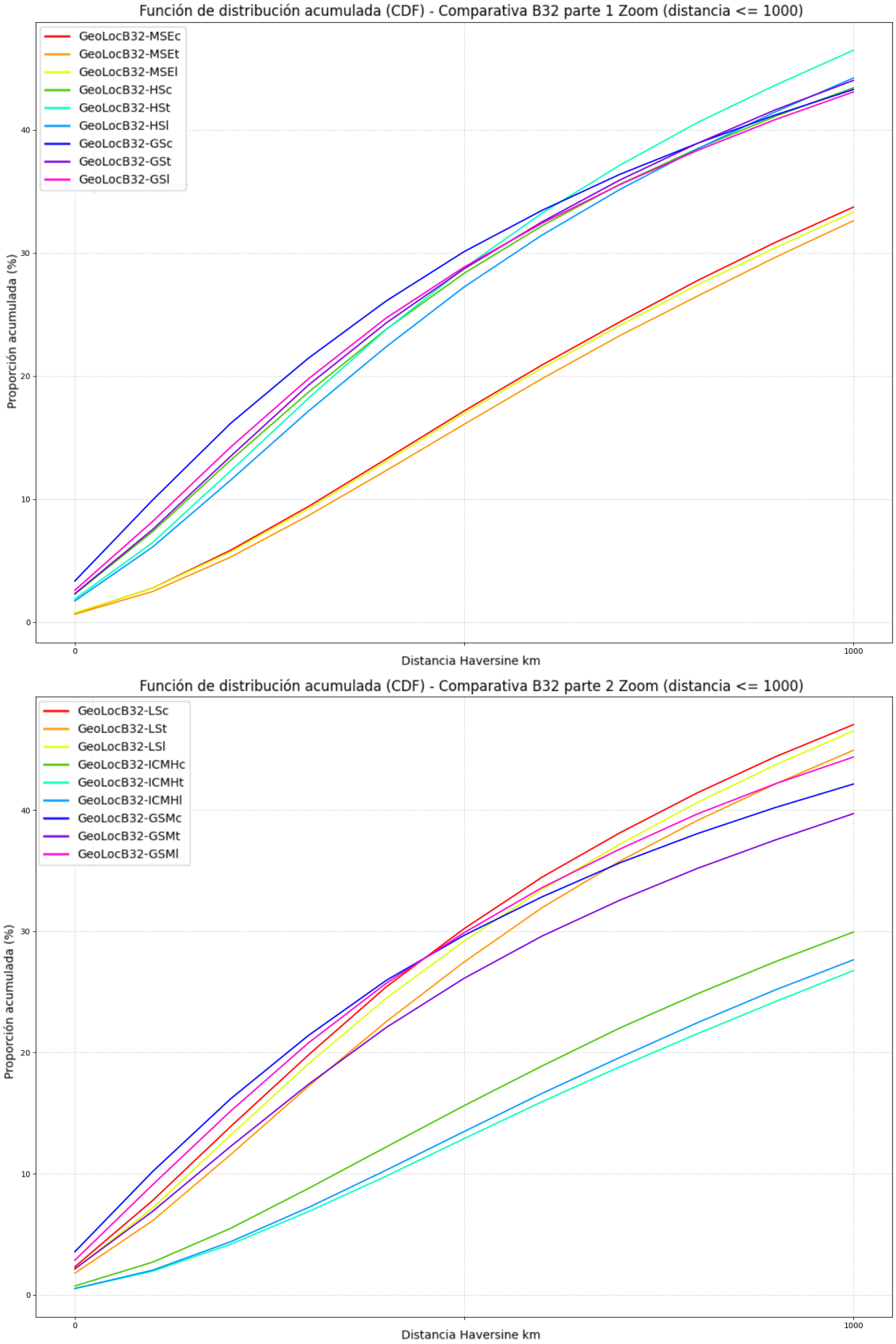

Figura B.19: Funciones de distribución acumulada (CDF) de todos los modelos B32 ZOOM (distancia ≤ 1000)

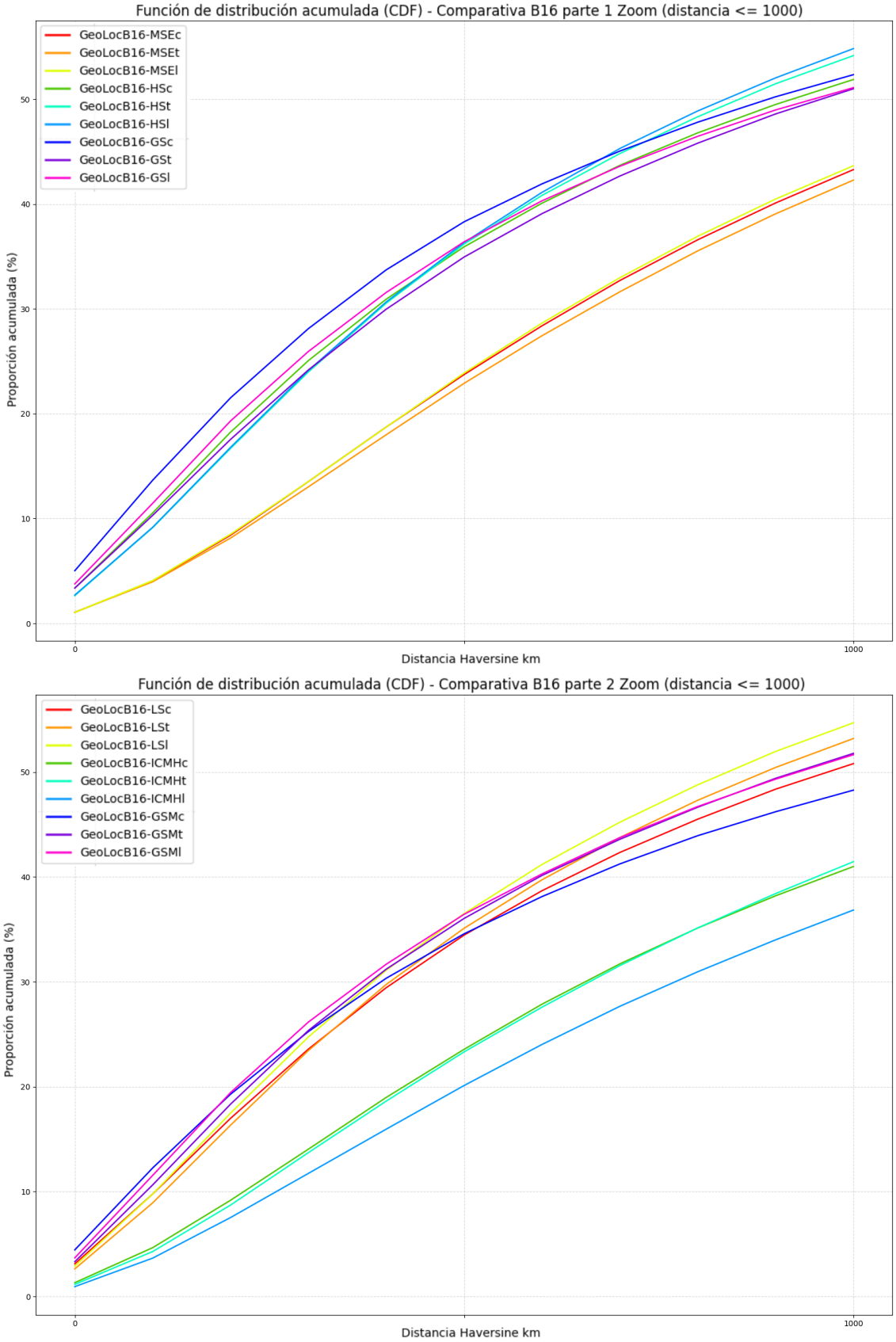

Figura B.20: Funciones de distribución acumulada (CDF) de todos los modelos B16 ZOOM (distancia ≤ 1000)

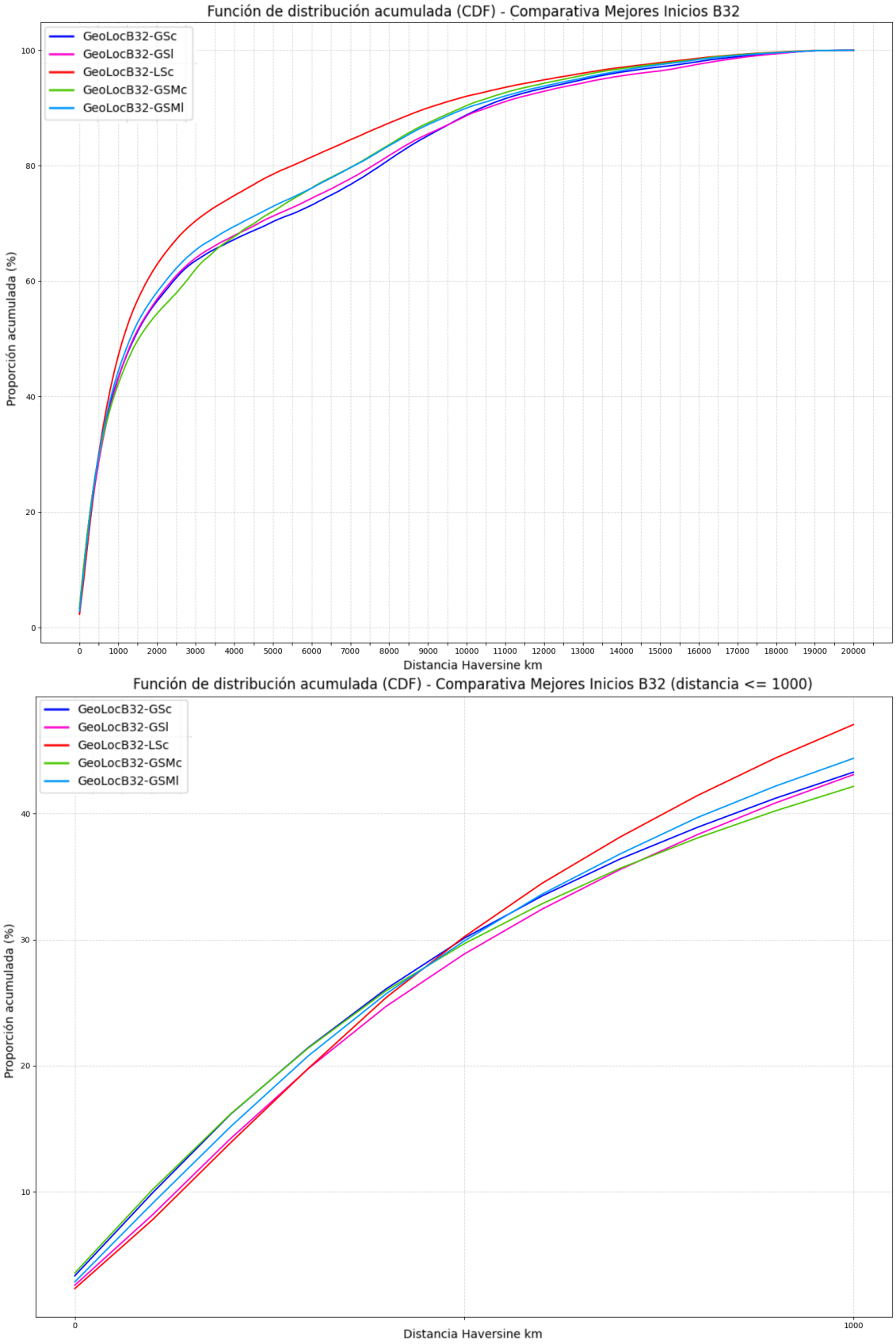

Figura B.21: Mejores inicios en las funciones de distribución acumulada de los modelos B32

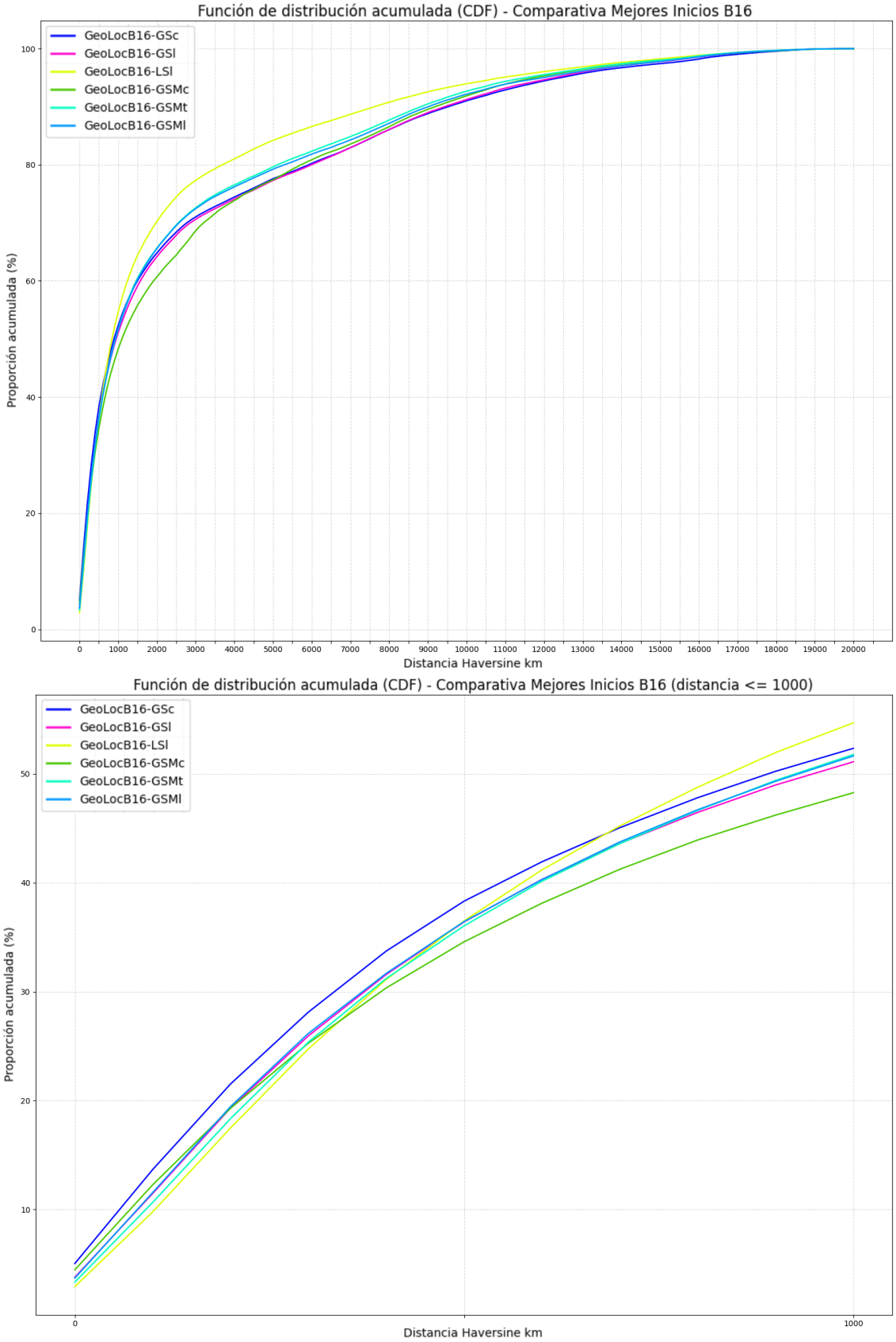

Figura B.22: Mejores inicios en las funciones de distribución acumulada de los modelos B16

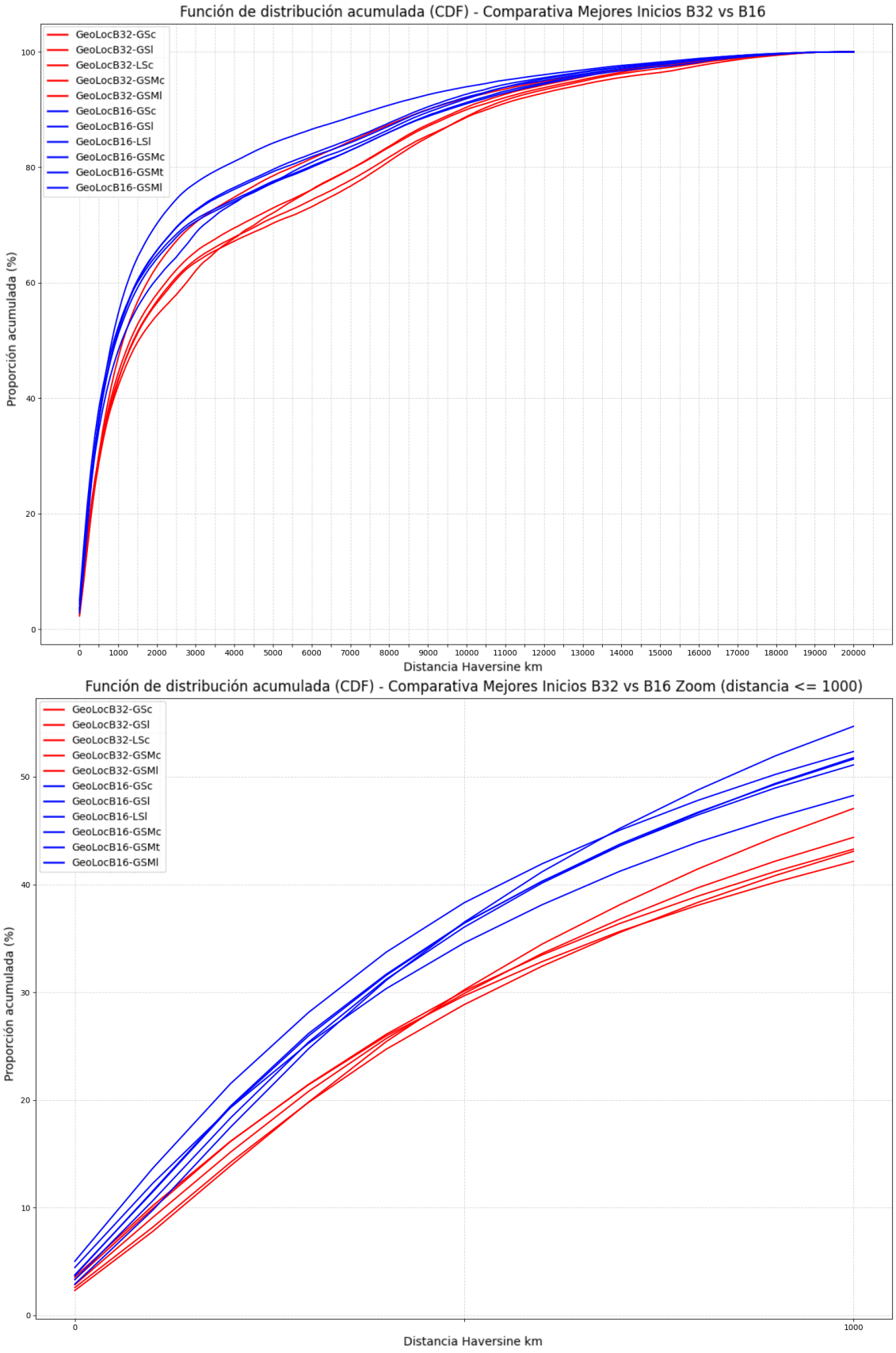

Figura B.23: Comparación mejores inicios en las funciones de distribución acumulada de los modelos B32 vs B16