



CentroGeo
19°17'30"N 99°13'17"W 2489m

Centro de Investigación en
Ciencias de Información Geoespacial

CentroGeo

Maestría en CIG

Aprendizaje Profundo

Proyecto Final

@date: 2024-08-15

@profesor: Dr. Rodrigo López Farías

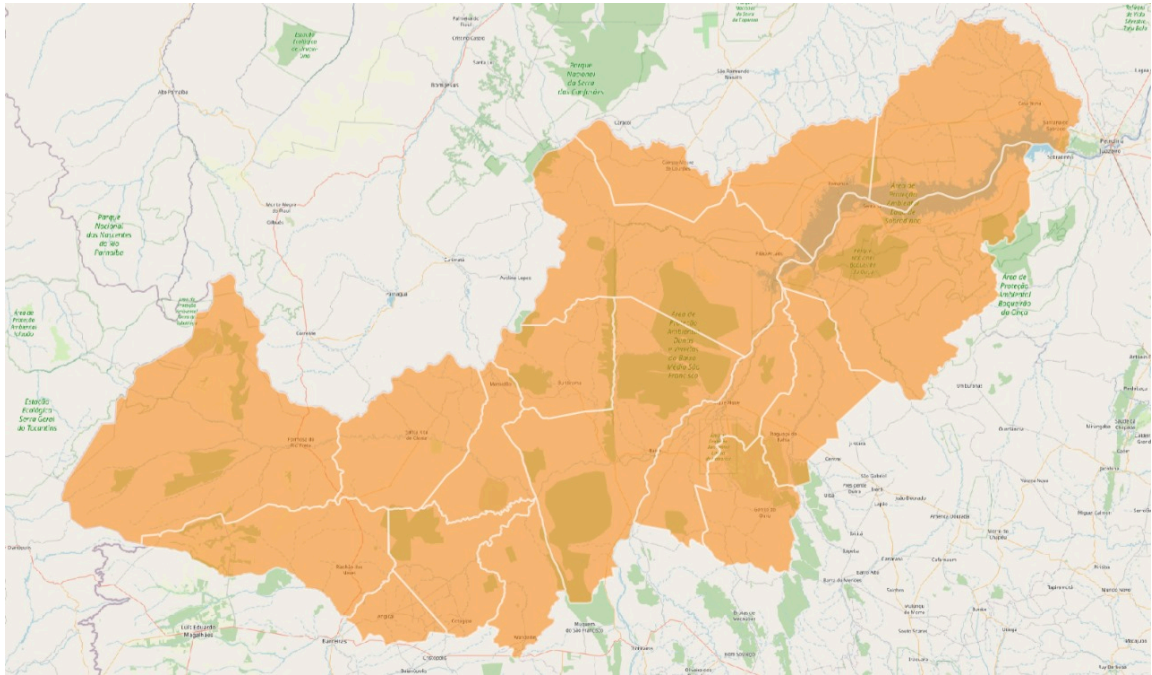
@autor: Uriel Mendoza; Raúl Sánchez

Predicción de la deforestación con una red convolucional basada en datos de 1985 a 2019, en la unidad federativa de Bahía, Brasil.

Introducción

Este trabajo presenta el desarrollo de un modelo basado en redes neuronales convolucionales (CNNs) para analizar y predecir la deforestación en los municipios de Angical, Barra, Buritirama, Campo Alegre de Lourdes, Casa Nova, Cotegipe, Formosa do Rio Preto, Gentio do Ouro, Itaguaçu da Bahia, Mansidão, Pilão Arcado, Remanso, Riachão das Neves, Santa Rita de Cássia, Sento Sé, Wanderley y Xique-Xique, en la unidad federativa de Bahía, Brasil.

Zona de estudio



A partir de datos geoespaciales (imágenes raster) que cubren los años de 1985 a 2020, se generan histogramas que representan cambios en la cobertura del suelo, basada en la distancia a la deforestación. Estos histogramas se utilizan para entrenar un modelo que pueda predecir la deforestación futura, en este caso, específicamente para el año 2020. El proyecto incluye la implementación de técnicas de validación cruzada para asegurar la robustez del modelo, así como la visualización del progreso del entrenamiento para evaluar su convergencia. Se emplearon dos enfoques diferentes: el método matricial, que organiza los histogramas en bloques temporales para capturar mejor las tendencias históricas, y el método normal, que trata cada año de manera independiente. Ambos enfoques se comparan para determinar cuál proporciona una mejor predicción del fenómeno estudiado.

Metodología

Importación de Bibliotecas

En esta sección, se importan las bibliotecas necesarias para la manipulación de datos raster, análisis de datos, construcción y entrenamiento de modelos de aprendizaje profundo, y visualización. Se incluye:

- rasterio: Para la manipulación de datos raster (imágenes geoespaciales).
- numpy: Para operaciones numéricas y manipulación de matrices.
- matplotlib: Para la visualización de datos.
- tensorflow: Para la creación y entrenamiento de modelos de aprendizaje profundo.
- scikit-learn: Para funciones relacionadas con el modelado estadístico, incluyendo la división de datos y validación cruzada.

```
In [ ]: # Asegúrate de instalar rasterio si no está disponible
!pip install rasterio
```

```
Requirement already satisfied: rasterio in /usr/local/lib/python3.10/dist-packages (1.3.10)
Requirement already satisfied: affine in /usr/local/lib/python3.10/dist-packages (from rasterio) (2.4.0)
Requirement already satisfied: attrs in /usr/local/lib/python3.10/dist-packages (from rasterio) (24.2.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from rasterio) (2024.7.4)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.10/dist-packages (from rasterio) (8.1.7)
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.10/dist-packages (from rasterio) (0.7.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from rasterio) (1.26.4)
Requirement already satisfied: snuggs>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from rasterio) (1.4.7)
Requirement already satisfied: click-plugins in /usr/local/lib/python3.10/dist-packages (from rasterio) (1.1.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from rasterio) (71.0.4)
Requirement already satisfied: pyparsing>=2.1.6 in /usr/local/lib/python3.10/dist-packages (from snuggs>=1.4.1->rasterio) (3.1.2)
```

```
In [ ]: import importlib
import rasterio
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split, KFold
import pandas as pd
```

```
In [ ]: # Importar módulos personalizados
import tocplus as tocp
```

Definiciones Iniciales

Aquí se definen algunos parámetros importantes para el análisis:

- yinit y yend: Definen el rango de años de los datos que se analizarán.
- nbins: El número de bins en el histograma, que determina la resolución de las distribuciones de frecuencias.
- LAGS: Número de pasos de tiempo previos utilizados en la predicción.
- HALF_SIZE: Parámetro utilizado para definir el tamaño de las ventanas de datos en análisis espacial.
- batch_size: Tamaño del lote para el entrenamiento de la red neuronal.
- epochs: Número de épocas de entrenamiento.

```
In [ ]: # Definiciones iniciales
yinit = 1985
```

```

yend = 2020
nbins = 35
LAGS = 3
HALF_SIZE = 11
batch_size = 25
epochs = 50

```

Función para Cargar los Datos y Generar Histogramas

La función `cargar_datos_y_generar_histogramas` procesa imágenes raster de datos de pérdida forestal para generar histogramas normalizados que luego se utilizarán en modelos predictivos.

Recibe indirectamente parámetros globales como `yinit` (año inicial), `yend` (año final), y `nbins` (número de bins). Carga una secuencia de imágenes raster $R(y)$, donde y varía de `yinit` a `yend`. Cada imagen $R(y)$ tiene dimensiones espaciales (h, w) , que corresponden a la altura y el ancho del raster. Se crea un tensor $D2NF$ de dimensiones (h, w, t) , donde $t = yend - yinit + 1$, para almacenar las imágenes raster de cada año en el tiempo.

Para cada imagen $R(y)$, se aplana la imagen en un vector \mathbf{r} de tamaño $h \times w$. Luego, se calcula un histograma de \mathbf{r} en `nbins` categorías y este histograma se almacena en una matriz H de dimensiones $(t - 1, nbins)$. La matriz H se normaliza para que la suma de cada fila sea igual a 1, es decir:

$$H[i, :] = \frac{H[i, :]}{\sum H[i, :]}$$

La función devuelve dos salidas principales: el tensor tridimensional `D2NF` de dimensiones (h, w, t) , que contiene las imágenes raster para cada año, y la matriz de histogramas normalizados `histograms` con dimensiones $(t - 1, nbins)$, donde cada fila corresponde a un año y cada columna a un bin del histograma. Esto permite estructurar la información espacial y temporal de los datos para su uso en modelos de predicción basados en aprendizaje profundo.

```

In [ ]: # Función para cargar los datos y generar histogramas
def cargar_datos_y_generar_histogramas():
    path_to_initial_raster = "/content/drive/MyDrive/CG_AP_proyectoFinal_RS_UM/data

    with rasterio.open(path_to_initial_raster) as src:
        img = src.read(1)
        sh = img.shape

        tensor = (sh[0], sh[1], yend - yinit + 1)
        D2NF = np.zeros(tensor)

        for i, y in enumerate(np.arange(yinit, yend + 1)):
            path_to_d2nf = "/content/drive/MyDrive/CG_AP_proyectoFinal_RS_UM/data/proce
            with rasterio.open(path_to_d2nf) as src:

```

```

        M = src.read(1)
        if M.shape == sh:
            D2NF[:, :, i] = M
        else:
            raise ValueError(f"Dimensiones del archivo {path_to_d2nf} no coinci

histograms = np.zeros((yend - yinit, nbins))
for i, j in enumerate(np.arange(yinit, yend)):
    rank = D2NF[:, :, i].flatten()
    ground_truth = D2NF[:, :, i + 1] == 0
    potential_deforestation_map_mask = ground_truth.flatten()
    X = rank[rank > 0]
    Y = potential_deforestation_map_mask[rank > 0] * 1
    T1 = tocp.TOC(rank=np.array(X), groundtruth=np.array(Y))
    unique_ranks = T1.TPplusFP[~T1.idiscontinuous[:]]
    unique_TP = T1.TP[~T1.idiscontinuous[:]]
    counts = unique_TP[1:] - unique_TP[:-1]
    histograms[i, :] = counts[:nbins]

# Normalizar los histogramas para que la suma sea 1
    histograms = histograms / histograms.sum(axis=1, keepdims=True)

return D2NF, histograms

```

Ploteo de imágenes

La función `mostrar_imagenes` visualiza secuencialmente las imágenes raster de datos de pérdida forestal para cada año dentro de un rango especificado.

La función recibe el parámetro `D2NF`, un tensor tridimensional de dimensiones (h, w, t) que contiene imágenes raster para cada año dentro del rango definido por `yinit` y `yend`. Este tensor se visualiza utilizando la biblioteca `matplotlib`.

Dentro de un lazo `for`, la función recorre cada capa del tensor (cada año) y muestra la imagen correspondiente utilizando la paleta de colores `viridis`. Cada imagen se coloca en una cuadrícula de subplots, específicamente en una matriz de 5 filas y 8 columnas (capaz de mostrar hasta 40 imágenes), y se etiqueta con el año correspondiente.

La función no devuelve ningún valor, pero muestra un gráfico en pantalla con todas las imágenes raster organizadas en una cuadrícula. Esto permite una inspección visual de los datos a lo largo del tiempo, facilitando la identificación de patrones y cambios temporales en la deforestación.

```

In [ ]: def mostrar_imagenes(D2NF):
        plt.figure(figsize=(20, 10))
        for i in range(D2NF.shape[2]):
            plt.subplot(5, 8, i+1)
            plt.imshow(D2NF[:, :, i], cmap='viridis') # Usar la paleta 'viridis'
            plt.title(f'Año {yinit + i}')
            plt.axis('off')

```

```
plt.tight_layout()
plt.show()
```

Graficación de histogramas

La función `graficar_histogramas` genera un gráfico que muestra los histogramas de frecuencias para cada año dentro de un rango especificado.

Esta función recibe como entrada el parámetro `histograms`, que es una matriz de dimensiones (t, b) , donde t es el número de años (diferencia entre `yend` y `yinit`), y b es el número de bins en cada histograma.

Dentro de un lazo `for`, la función recorre cada fila de la matriz `histograms`, correspondiente a un año específico, y grafica los valores de cada histograma en el mismo gráfico. El eje x del gráfico representa los bins, y el eje y representa la frecuencia para cada bin.

Se agregan etiquetas de los ejes, una leyenda para identificar el año correspondiente a cada histograma, y un título para el gráfico. Al finalizar, se muestra el gráfico que permite comparar visualmente cómo varían las distribuciones de frecuencias a lo largo del tiempo.

Esta función no devuelve ningún valor, pero genera y muestra un gráfico en pantalla que facilita la comparación y análisis de los histogramas de deforestación para todos los años considerados.

```
In [ ]: def graficar_histogramas(histograms):
plt.figure(figsize=(12, 8))
# Seleccionar una paleta de colores con buen contraste
colors = plt.cm.get_cmap('tab20', histograms.shape[0])
# Graficar cada histograma con un color distinto
for i in range(histograms.shape[0]):
    plt.plot(histograms[i, :], label=f'{yinit + i}', color=colors(i))
plt.xlabel('Bins')
plt.ylabel('Frecuencia')
# Mostrar todos los años en la leyenda y organizarla en varias columnas
plt.legend(title="Años", bbox_to_anchor=(1.05, 1), loc='upper left', ncol=2) #
plt.title('Histogramas Normalizados de Frecuencias de Todos los Años')
plt.tight_layout()
plt.show()
```

Graficación individual de histogramas

La función `graficar_histogramas_individuales` se utiliza para generar gráficos individuales de los histogramas de cada año en un formato de cuadrícula, permitiendo una comparación visual clara y detallada de las distribuciones de frecuencia año por año.

Esta función recibe como entrada el parámetro `histograms`, que es una matriz de dimensiones (t, b) , donde t es el número de años (desde `yinit` hasta `yend`), y b es el

número de bins en cada histograma.

Dentro de la función, se crea una cuadrícula de subgráficos (6x6) para alojar los histogramas de cada año. Luego, mediante un lazo `for`, se recorre cada histograma en la matriz `histograms`, y se grafica en su subgráfico correspondiente utilizando un gráfico de barras.

Cada subgráfico incluye un título que indica el año correspondiente al histograma mostrado. Finalmente, se ajusta el espaciado entre los gráficos para evitar superposiciones y se muestra la cuadrícula completa.

Esta función no devuelve ningún valor, pero genera y muestra en pantalla una cuadrícula de gráficos, donde cada gráfico corresponde al histograma de un año específico, facilitando la comparación visual directa de los datos año tras año.

```
In [ ]: def graficar_histogramas_individuales(histograms):
    fig, axs = plt.subplots(6, 6, figsize=(15, 15))
    axs = axs.flatten()
    for i in range(histograms.shape[0]):
        axs[i].bar(np.arange(nbins), histograms[i, :])
        axs[i].set_title(f"Histograma {yinit + i}")
    plt.tight_layout()
    plt.show()
```

Construcción de la Matriz de Histogramas

La función `construir_matriz_histogramas` reorganiza una secuencia de histogramas anuales en bloques de 4 años consecutivos, agrupándolos en una matriz tridimensional. Cada fila de esta matriz contiene 4 histogramas consecutivos, lo que permite capturar la evolución temporal de los datos en esos periodos.

La función toma una serie de histogramas (uno por año) y crea una nueva matriz tridimensional donde cada fila es un conjunto de 4 histogramas consecutivos. La idea es analizar cómo cambian los datos a lo largo del tiempo en bloques de 4 años.

En notación matemática, si h_i representa el histograma del año i , la matriz resultante M se vería así:

$$M = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 \\ h_2 & h_3 & h_4 & h_5 \\ h_3 & h_4 & h_5 & h_6 \\ \vdots & \vdots & \vdots & \vdots \\ h_{n-3} & h_{n-2} & h_{n-1} & h_n \end{bmatrix}$$

Donde n es el número total de años menos 3, y cada h_i es un vector que representa el histograma para el año i . Así, cada fila de la matriz M contiene 4 histogramas consecutivos, que pueden ser usados para análisis o entrenamiento de modelos que consideran la dinámica temporal de los datos.

```
In [ ]: # Función para construir la matriz de histogramas para el entrenamiento y la validación
def construir_matriz_histogramas(histogramas):
    filas = histogramas.shape[0] - 3 # Restamos 3 porque vamos a formar bloques de
    columnas = 4
    matriz_histogramas = np.zeros((filas, columnas, nbins))

    for i in range(filas):
        matriz_histogramas[i, 0, :] = histogramas[i, :]
        matriz_histogramas[i, 1, :] = histogramas[i + 1, :]
        matriz_histogramas[i, 2, :] = histogramas[i + 2, :]
        matriz_histogramas[i, 3, :] = histogramas[i + 3, :]

    return matriz_histogramas
```

División de Datos de Histogramas

La función `dividir_datos` permite dividir los datos de histogramas en conjuntos de entrenamiento y validación, utilizando dos modos diferentes: "normal" y "matricial". Dependiendo del valor del parámetro `modo`, la función organiza y divide los histogramas de manera diferente.

En el modo 'normal', los histogramas se dividen en datos de entrada y etiquetas, separándolos en conjuntos de entrenamiento y validación según un porcentaje.

Para un conjunto de histogramas h_i , los datos se organizan de la siguiente manera:

$$X = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{n-1} \end{bmatrix}, \quad Y = \begin{bmatrix} h_2 \\ h_3 \\ \vdots \\ h_n \end{bmatrix}$$

Aquí, X contiene todos los histogramas menos el último, mientras que Y contiene todos los histogramas desde el segundo hasta el último.

En el modo 'matricial', los datos se reorganizan en bloques de 4 histogramas consecutivos. De cada bloque, los primeros 3 histogramas se utilizan como entrada y el cuarto como etiqueta.

En notación matemática, si tenemos histogramas $h_i, h_{i+1}, h_{i+2}, h_{i+3}$, la matriz resultante se verá así:

$$X = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_2 & h_3 & h_4 \\ \vdots & \vdots & \vdots \\ h_{n-3} & h_{n-2} & h_{n-1} \end{bmatrix}, \quad Y = \begin{bmatrix} h_4 \\ h_5 \\ \vdots \\ h_n \end{bmatrix}$$

Aquí, cada fila de la matriz X representa un bloque de 3 histogramas consecutivos, y la matriz Y contiene el histograma siguiente en la secuencia temporal.

```
In [ ]: # Función para dividir los datos
def dividir_datos(histogramas, modo='normal', porcentaje_entrenamiento=0.8):
    if modo == 'normal':
        split_index = int(histogramas.shape[0] * porcentaje_entrenamiento)
        X = histogramas[:-1, :].reshape((-1, nbins, 1, 1)) # Datos de entrenamiento
        Y = histogramas[1:, :].reshape((-1, nbins)) # Histograma de 2020 como etiqueta

        # Dividir en conjunto de entrenamiento y validación según los años
        X_train, X_val = X[:split_index], X[split_index:]
        Y_train, Y_val = Y[:split_index], Y[split_index:]

        # Imprime el tamaño de la matriz
        print("Tamaño de la matriz X_train:", X_train.shape)
        print("Tamaño de la matriz X_val:", X_val.shape)
        print("Tamaño de la matriz Y_train:", Y_train.shape)
        print("Tamaño de la matriz Y_val:", Y_val.shape)

    elif modo == 'matricial':
        matriz_histogramas = construir_matriz_histogramas(histogramas)
        X = matriz_histogramas[:, :3, :].reshape((-1, 3, nbins, 1)) # Las primeras
        Y = matriz_histogramas[:, 3, :] # La última columna como etiqueta

        # No es necesario dividir aquí porque ya estamos definiendo los datos de va
        X_train, X_val = X, X
        Y_train, Y_val = Y, Y

        # Imprime el tamaño de la matriz
        print("Tamaño de la matriz X_train:", X_train.shape)
        print("Tamaño de la matriz X_val:", X_val.shape)
        print("Tamaño de la matriz Y_train:", Y_train.shape)
        print("Tamaño de la matriz Y_val:", Y_val.shape)

    return X_train, X_val, Y_train, Y_val
```

Definición del Modelo de Red Neuronal

La función `definir_modelo` construye y define un modelo de red neuronal convolucional utilizando TensorFlow. Esta función está diseñada para trabajar en dos modos diferentes, dependiendo de cómo se estructuren los datos de entrada: el modo 'normal' y el modo 'matricial'.

En el modo 'normal', los datos de entrada son histogramas individuales, por lo que la forma de entrada es `(nbins, 1, 1)`, donde `nbins` es el número de bins en cada histograma.

En el modo 'matricial', los datos de entrada consisten en bloques de 3 histogramas consecutivos, por lo que la forma de entrada es `(3, nbins, 1)`.

Matemáticamente, si un histograma tiene una dimensión n , la entrada en el modo 'normal' es:

$$\text{input_shape} = (n, 1, 1)$$

Y en el modo 'matricial' es:

$$\text{input_shape} = (3, n, 1)$$

La función construye un modelo secuencial (`Sequential`), que es una pila lineal de capas. La primera capa es una capa convolucional 2D con 16 filtros, un tamaño de kernel de `(3, 1)`, y una activación ReLU. El padding se establece en `same`, lo que significa que la salida tendrá la misma dimensión espacial que la entrada.

Después de la capa convolucional, se agrega una capa de `MaxPooling2D` que reduce la dimensión espacial de la salida a la mitad en la primera dimensión.

Luego, se añade otra capa convolucional 2D con 32 filtros.

La salida de la segunda capa convolucional se aplanan (`Flatten`), convirtiendo los datos en un vector de una dimensión.

A continuación, se agrega una capa densa con 64 unidades y activación ReLU.

Finalmente, una capa densa con `nbins` unidades se utiliza para la predicción directa de los valores del histograma.

En términos de arquitectura, el modelo se puede representar como:

$$\text{Model} = \text{Flatten}(\text{Conv2D}(\text{MaxPooling2D}(\text{Conv2D}(\text{Input}))))$$

El modelo se compila con el optimizador `adam`, que es un método de optimización estocástica. La función de pérdida utilizada es el error cuadrático medio (`mse`), una medida de la diferencia promedio cuadrada entre los valores predichos y los valores reales. La métrica utilizada para evaluar el rendimiento es el error absoluto medio (`mae`).

En el modo 'normal':

$$\text{input_shape} = (n, 1, 1)$$

donde n es el número de bins en cada histograma.

En el modo 'matricial':

`input_shape = (3, n, 1)`

donde 3 es el número de histogramas consecutivos considerados y n es el número de bins en cada histograma.

```
In [ ]: # Función para definir el modelo
def definir_modelo(modo='normal'):
    if modo == 'normal':
        input_shape = (nbins, 1, 1)
    elif modo == 'matricial':
        input_shape = (3, nbins, 1)

    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=input_shape),
        tf.keras.layers.Conv2D(16, (3, 1), activation='relu', padding='same'),
        tf.keras.layers.MaxPooling2D((2, 1)),
        tf.keras.layers.Conv2D(32, (3, 1), activation='relu', padding='same'),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(nbins, activation='linear') # Predicción directa de
    ])

    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model
```

Entrenamiento y Validación del Modelo

La función `entrenar_y_validar` se encarga de entrenar un modelo de red neuronal utilizando un procedimiento de validación cruzada con el objetivo de evaluar su rendimiento de manera más robusta. La validación cruzada es una técnica que divide los datos de entrenamiento en múltiples subconjuntos (o folds) para asegurarse de que el modelo se evalúe de manera consistente en diferentes particiones de los datos.

La función utiliza `KFold` de `sklearn.model_selection`, que divide el conjunto de datos de entrenamiento en 5 pliegues o folds, donde cada pliegue actúa como conjunto de validación una vez mientras que los restantes pliegues actúan como conjunto de entrenamiento. La variable `fold_no` lleva el seguimiento del número de pliegues (folds) que se están entrenando y validando.

Dentro del bucle `for`, los datos de entrenamiento y validación para cada pliegue se seleccionan utilizando los índices proporcionados por `KFold`. Para cada pliegue, se define un nuevo modelo llamando a la función `definir_modelo`, que configura la arquitectura de la red neuronal en función del modo (normal o matricial). Se entrena el modelo utilizando el método `fit`, pasando los datos de entrenamiento y validación correspondientes a ese pliegue. Durante el entrenamiento, la función `plot_history` se llama para graficar el progreso del modelo, mostrando cómo evolucionan el error de entrenamiento y el error de validación en cada época.

Después de completar todos los pliegues, la función devuelve el modelo entrenado. Este modelo ha sido ajustado y evaluado en cada pliegue de validación, lo que proporciona una estimación más confiable del rendimiento general del modelo.

Matemáticamente, la validación cruzada puede representarse como:

$$\text{Errores} = \frac{1}{K} \sum_{k=1}^K \text{Error}_k$$

donde K es el número de pliegues, y Error_k es el error en el k -ésimo pliegue.

En cada iteración del bucle, se selecciona un subconjunto de datos para el entrenamiento $X_{\text{train}}^{(k)}$, $Y_{\text{train}}^{(k)}$ y un subconjunto para la validación $X_{\text{val}}^{(k)}$, $Y_{\text{val}}^{(k)}$. Luego, el modelo se entrena en $X_{\text{train}}^{(k)}$ y se evalúa en $X_{\text{val}}^{(k)}$.

El ciclo se repite para K pliegues, y finalmente, el rendimiento general del modelo se evalúa en función del promedio de los errores obtenidos en cada pliegue.

```
In [ ]: # Función para entrenar el modelo y realizar validación cruzada
def entrenar_y_validar(X_train, Y_train, modo='normal'):
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    fold_no = 1
    for train_index, val_index in kfold.split(X_train):
        print(f"Entrenando fold {fold_no}...")
        X_tr, X_vl = X_train[train_index], X_train[val_index]
        Y_tr, Y_vl = Y_train[train_index], Y_train[val_index]
        model = definir_modelo(modo=modo)
        history = model.fit(X_tr, Y_tr, epochs=epochs, batch_size=batch_size, valid
        fold_no += 1
        plot_history(history)

    return model
```

Visualización del Progreso del Entrenamiento

La función `plot_history` se utiliza para visualizar el progreso del entrenamiento de un modelo de red neuronal a lo largo de las épocas. Esta función toma como entrada el objeto `history` generado durante el entrenamiento del modelo, el cual contiene los valores de las métricas de rendimiento (como el error absoluto medio, `mae`, y la pérdida) tanto en los datos de entrenamiento como en los datos de validación para cada época.

Primero, los datos históricos de entrenamiento se convierten en un DataFrame de pandas para facilitar la manipulación y el acceso a las métricas. Luego, se añade una columna al DataFrame que corresponde al número de época.

La función luego crea una gráfica que muestra cómo el error absoluto medio (`mae`) evoluciona a lo largo del tiempo tanto para el conjunto de entrenamiento como para el de validación. En el eje X se muestran las épocas y en el eje Y se representa el error absoluto

medio. Se trazan dos curvas: una para el error en los datos de entrenamiento (**Train Error**) y otra para el error en los datos de validación (**Val Error**).

Esta visualización es crucial para analizar el comportamiento del modelo durante el entrenamiento. Permite identificar problemas como el sobreajuste, que se manifiesta cuando el error en los datos de entrenamiento sigue disminuyendo mientras que el error en los datos de validación se estabiliza o incluso aumenta.

La gráfica producida por **plot_history** proporciona una visión clara de cómo el modelo está aprendiendo y si está generalizando bien a los datos no vistos durante el entrenamiento.

```
In [ ]: # Función para graficar la historia de entrenamiento
def plot_history(history):
    hist = pd.DataFrame(history.history)
    hist['epoch'] = history.epoch

    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Mean Abs Error')
    plt.plot(hist['epoch'], hist['mae'], label='Train Error')
    plt.plot(hist['epoch'], hist['val_mae'], label='Val Error')
    plt.legend()
    plt.show()
```

Predicción y Gráficas con el Modelo Entrenado

La función **predecir_y_graficar** se utiliza para hacer predicciones con el modelo entrenado y comparar esas predicciones con los valores reales, representados en un histograma. Esta función recibe como parámetros el modelo entrenado, los datos de validación, los histogramas originales y un parámetro **modo** que indica si se están usando los datos en su forma normal o matricial.

Primero, dependiendo del **modo**, la función prepara los datos de entrada para la predicción:

- Si el **modo** es **'normal'**, la función toma el último conjunto de datos de validación (**X_val[-1]**), lo reconfigura en un formato adecuado para el modelo (una matriz de dimensión **(1, nbins, 1, 1)**), y lo usa para predecir.
- Si el **modo** es **'matricial'**, el último conjunto de datos de validación se reconfigura en una matriz tridimensional con dimensiones **(1, 3, nbins, 1)** antes de hacer la predicción.

Luego, el modelo hace la predicción sobre el conjunto de datos ajustado. Esta predicción es un histograma que representa las frecuencias estimadas para el año 2020.

La función finaliza generando una gráfica de barras para comparar las predicciones del modelo con los valores reales del histograma del año 2020. En la gráfica, las barras que

representan las predicciones se superponen a las barras que muestran los valores reales, lo que permite una fácil comparación visual entre las dos.

Esta comparación gráfica es esencial para evaluar cómo de bien el modelo puede predecir las distribuciones de frecuencias de los histogramas para años que no se han utilizado en el entrenamiento, proporcionando una evaluación clara de la capacidad predictiva del modelo.

```
In [ ]: # Función para predecir y graficar resultados
def predecir_y_graficar(model, X_val, histograms, modo='normal'):
    if modo == 'normal':
        X_2020 = X_val[-1].reshape(1, nbins, 1, 1)
    elif modo == 'matricial':
        X_2020 = X_val[-1].reshape(1, 3, nbins, 1)

    prediction = model.predict(X_2020)

    fig, ax = plt.subplots()
    ax.bar(np.arange(nbins), prediction.flatten(), label="Predicted 2020")
    ax.bar(np.arange(nbins), histograms[-1, :], alpha=0.3, label="Actual 2020")
    ax.set_title("Histograma de frecuencias comparado")
    ax.set_xlabel("Bins")
    ax.set_ylabel("Frecuencia")
    plt.legend()
    plt.show()
```

Ejecución de funciones iniciales

La parte donde se ejecuta la función `cargar_datos_y_generar_histogramas()` para obtener dos variables importantes: `D2NF` y `histograms`.

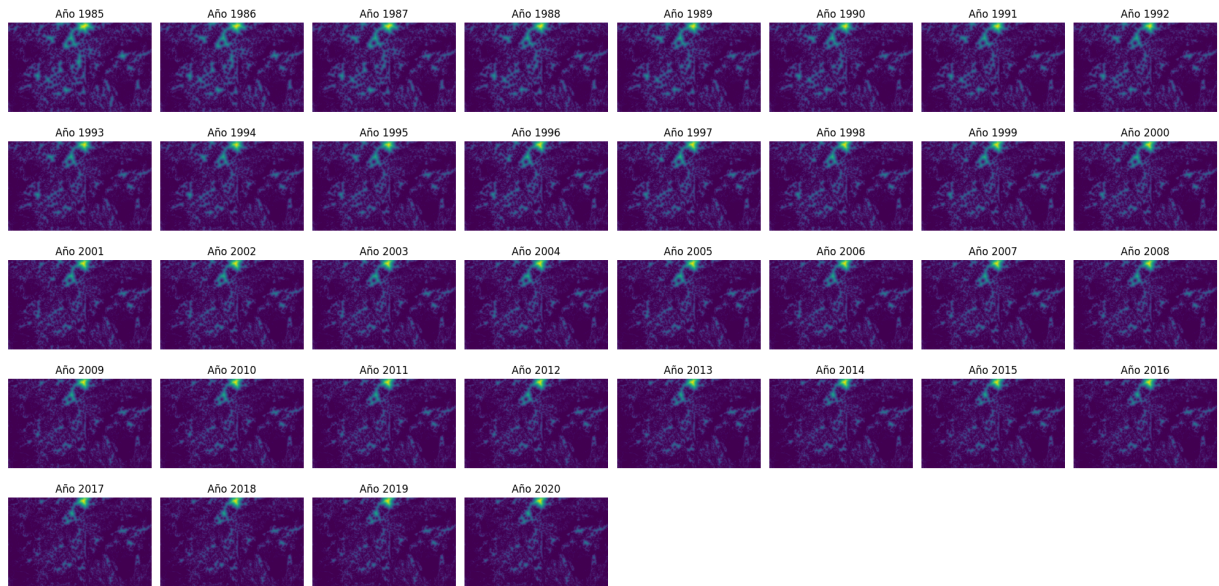
- `D2NF`: Es una matriz tridimensional donde se almacenan los datos rasterizados de deforestación a lo largo del tiempo. Cada "capa" de la matriz representa un año, desde 1985 hasta 2020, y contiene la información espacial de deforestación para ese año en particular.
- `histograms`: Es una matriz donde cada fila representa un año y cada columna representa un "bin" en un histograma. Los valores en esta matriz corresponden a la frecuencia de ciertos valores de deforestación (por ejemplo, áreas afectadas) dentro de los datos espaciales para cada año. Además, los histogramas han sido normalizados para que la suma de las frecuencias en cada año sea 1, lo cual facilita el análisis comparativo entre años.

```
In [ ]: # Ejecución del código
D2NF, histograms = cargar_datos_y_generar_histogramas()
```

La función `mostrar_imagenes(D2NF)` se encarga de mostrar visualmente las imágenes de los datos de deforestación para cada año dentro del rango especificado (1985-2020).

La función crea una figura grande y muestra cada capa de la matriz `D2NF` como una imagen, aplicando la paleta de colores `viridis` y etiquetando cada imagen con el año correspondiente. Esta visualización ayuda a inspeccionar rápidamente los patrones espaciales y temporales en los datos de deforestación.

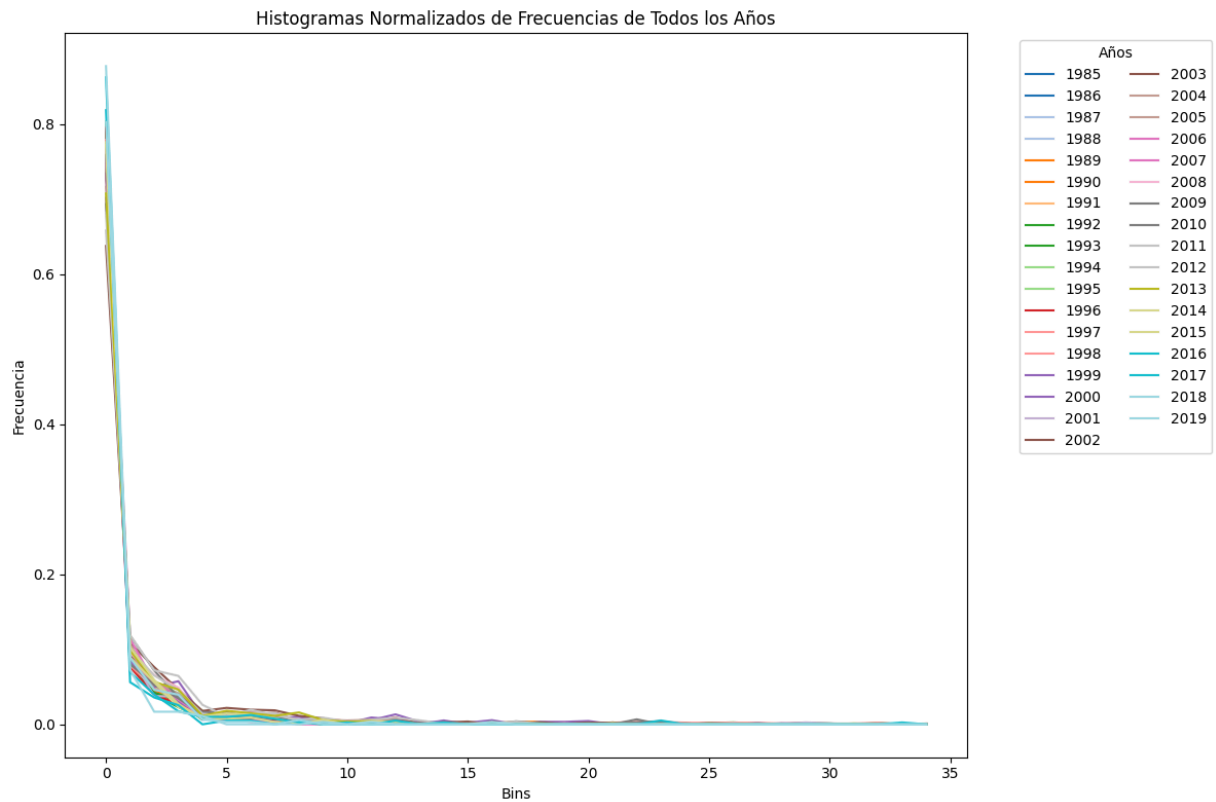
```
In [ ]: # Mostrar imágenes y gráficos de histogramas
mostrar_imagenes(D2NF)
```



La función `graficar_histogramas(histograms)` grafica los histogramas de frecuencias normalizadas para todos los años en el rango 1985-2020. La función toma el array `histograms`, que contiene los histogramas de cada año, y traza cada uno en una sola gráfica para facilitar la comparación. Esto permite visualizar cómo han cambiado las distribuciones de frecuencias de los datos de deforestación a lo largo del tiempo.

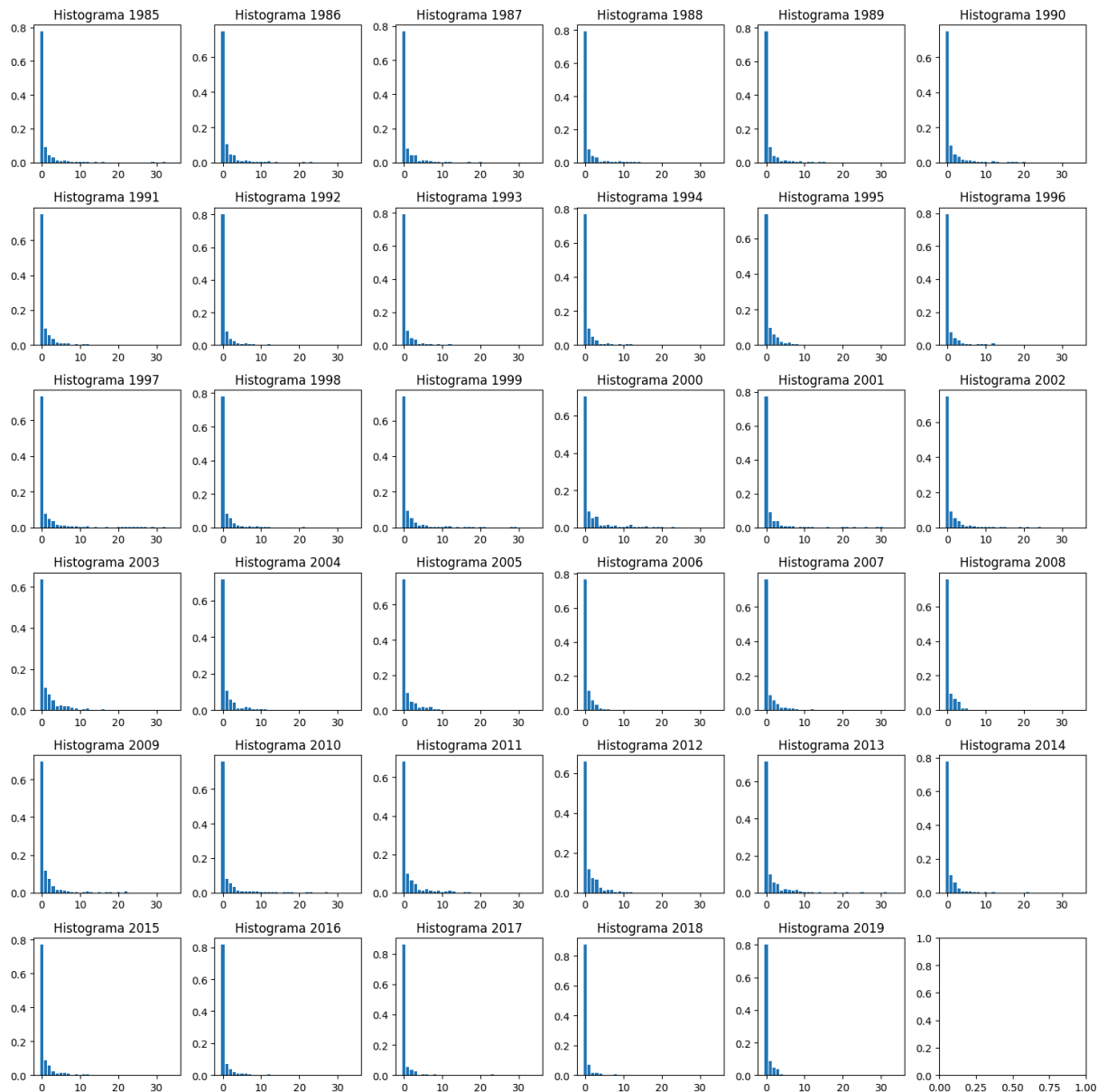
```
In [ ]: # Graficar histogramas normalizados de todos los años.
graficar_histogramas(histograms)
```

```
<ipython-input-58-34bc33e07e0d>:4: MatplotlibDeprecationWarning: The get_cmap function
was deprecated in Matplotlib 3.7 and will be removed two minor releases later. Use
`matplotlib.colormaps[name]` or `matplotlib.colormaps.get_cmap(obj)` instead.
  colors = plt.cm.get_cmap('tab20', histograms.shape[0])
```



La función `graficar_histogramas_individuales(histograms)` se encarga de generar gráficos individuales de los histogramas de cada año, mostrando cada uno en un subplot dentro de una figura grande. La función organiza los histogramas en una cuadrícula de 6x6, donde cada subplot muestra un histograma correspondiente a un año específico. Los gráficos permiten visualizar cómo se distribuyen los datos en cada año de manera individual dentro de una única imagen.

```
In [ ]: graficar_histogramas_individuales(histograms)
```

Modo matricial

La línea `modo = 'matricial'` establece la variable `modo` con el valor `'matricial'`, indicando que las funciones subsecuentes deben operar en este modo. En el contexto del código, esto significa que las funciones de procesamiento de datos, definición del modelo y entrenamiento se ajustarán para manejar los datos en formato matricial, donde se utilizan matrices de histogramas organizadas por bloques de tiempo.

```
In [ ]: # Elegir el modo: 'normal' o 'matricial'
modo = 'matricial'
```

La función `dividir_datos` organiza y divide los histogramas en conjuntos de entrenamiento y validación según el modo seleccionado ('normal' o 'matricial'). En el modo `'normal'`, simplemente separa los datos temporalmente. En el modo `'matricial'`, organiza los histogramas en bloques de tiempo, usando las primeras tres columnas como

entrada y la cuarta como etiqueta. Luego, devuelve estos conjuntos listos para el entrenamiento del modelo.

```
In [ ]: # Dividir los datos y entrenar el modelo
X_train, X_val, Y_train, Y_val = dividir_datos(histograms, modo=modo, porcentaje_en
```

Tamaño de la matriz X_train: (32, 3, 35, 1)

Tamaño de la matriz X_val: (32, 3, 35, 1)

Tamaño de la matriz Y_train: (32, 35)

Tamaño de la matriz Y_val: (32, 35)

Esta línea de código ejecuta la función `entrenar_y_validar`, la cual entrena el modelo utilizando validación cruzada. El parámetro `modo` determina si se entrenará el modelo en el modo `'normal'` o `'matricial'`. La función devuelve el modelo entrenado listo para hacer predicciones o evaluaciones posteriores.


```
In [ ]: # Entrena y valida el modelo, con validacion cruzada
model = entrenar_y_validar(X_train, Y_train, modo=modo)
```

Entrenando fold 1...


Epoch 1/50

1/1  5s 5s/step - loss: 0.0164 - mae: 0.0395 - val_loss: 0.0159
- val_mae: 0.0379


Epoch 2/50

1/1  0s 311ms/step - loss: 0.0157 - mae: 0.0379 - val_loss: 0.0151 - val_mae: 0.0370


Epoch 3/50

1/1  0s 89ms/step - loss: 0.0149 - mae: 0.0370 - val_loss: 0.0144 - val_mae: 0.0369


Epoch 4/50

1/1  0s 151ms/step - loss: 0.0141 - mae: 0.0369 - val_loss: 0.0135 - val_mae: 0.0377


Epoch 5/50

1/1  0s 99ms/step - loss: 0.0133 - mae: 0.0376 - val_loss: 0.0127 - val_mae: 0.0393


Epoch 6/50

1/1  0s 58ms/step - loss: 0.0125 - mae: 0.0393 - val_loss: 0.0119 - val_mae: 0.0412


Epoch 7/50

1/1  0s 56ms/step - loss: 0.0117 - mae: 0.0413 - val_loss: 0.0111 - val_mae: 0.0431


Epoch 8/50

1/1  0s 70ms/step - loss: 0.0109 - mae: 0.0432 - val_loss: 0.0102 - val_mae: 0.0448


Epoch 9/50

1/1  0s 72ms/step - loss: 0.0100 - mae: 0.0450 - val_loss: 0.0093 - val_mae: 0.0465


Epoch 10/50

1/1  0s 64ms/step - loss: 0.0091 - mae: 0.0466 - val_loss: 0.0085 - val_mae: 0.0481


Epoch 11/50

1/1  0s 136ms/step - loss: 0.0083 - mae: 0.0483 - val_loss: 0.0077 - val_mae: 0.0498


Epoch 12/50

1/1  0s 60ms/step - loss: 0.0075 - mae: 0.0499 - val_loss: 0.0070 - val_mae: 0.0512


Epoch 13/50

1/1  0s 57ms/step - loss: 0.0068 - mae: 0.0513 - val_loss: 0.0063 - val_mae: 0.0520


Epoch 14/50

1/1  0s 69ms/step - loss: 0.0062 - mae: 0.0522 - val_loss: 0.0058 - val_mae: 0.0519


Epoch 15/50

1/1  0s 58ms/step - loss: 0.0056 - mae: 0.0520 - val_loss: 0.0052 - val_mae: 0.0511


Epoch 16/50

1/1  0s 63ms/step - loss: 0.0051 - mae: 0.0512 - val_loss: 0.0047 - val_mae: 0.0496


Epoch 17/50


1/1  0s 60ms/step - loss: 0.0045 - mae: 0.0497 - val_loss: 0.0042 - val_mae: 0.0467


Epoch 18/50


1/1  0s 56ms/step - loss: 0.0040 - mae: 0.0467 - val_loss: 0.0036 - val_mae: 0.0428


Epoch 19/50


1/1  0s 59ms/step - loss: 0.0034 - mae: 0.0428 - val_loss: 0.003
1 - val_mae: 0.0389
Epoch 20/50


1/1  0s 60ms/step - loss: 0.0029 - mae: 0.0387 - val_loss: 0.002
6 - val_mae: 0.0357
Epoch 21/50


1/1  0s 67ms/step - loss: 0.0024 - mae: 0.0355 - val_loss: 0.002
1 - val_mae: 0.0323
Epoch 22/50


1/1  0s 74ms/step - loss: 0.0020 - mae: 0.0320 - val_loss: 0.001
7 - val_mae: 0.0287
Epoch 23/50


1/1  0s 130ms/step - loss: 0.0016 - mae: 0.0283 - val_loss: 0.00
14 - val_mae: 0.0253
Epoch 24/50


1/1  0s 67ms/step - loss: 0.0012 - mae: 0.0248 - val_loss: 0.001
1 - val_mae: 0.0231
Epoch 25/50


1/1  0s 60ms/step - loss: 9.6541e-04 - mae: 0.0224 - val_loss:
9.3986e-04 - val_mae: 0.0217
Epoch 26/50


1/1  0s 57ms/step - loss: 7.6992e-04 - mae: 0.0209 - val_loss:
7.9317e-04 - val_mae: 0.0204
Epoch 27/50


1/1  0s 56ms/step - loss: 6.2161e-04 - mae: 0.0195 - val_loss:
6.7448e-04 - val_mae: 0.0187
Epoch 28/50


1/1  0s 136ms/step - loss: 5.0063e-04 - mae: 0.0176 - val_loss:
5.7522e-04 - val_mae: 0.0171
Epoch 29/50


1/1  0s 138ms/step - loss: 3.9860e-04 - mae: 0.0158 - val_loss:
4.9660e-04 - val_mae: 0.0159
Epoch 30/50


1/1  0s 136ms/step - loss: 3.1706e-04 - mae: 0.0144 - val_loss:
4.5054e-04 - val_mae: 0.0148
Epoch 31/50


1/1  0s 70ms/step - loss: 2.6796e-04 - mae: 0.0133 - val_loss:
4.3722e-04 - val_mae: 0.0145
Epoch 32/50


1/1  0s 69ms/step - loss: 2.5135e-04 - mae: 0.0128 - val_loss:
4.4047e-04 - val_mae: 0.0143
Epoch 33/50














1/1  0s 151ms/step - loss: 2.5113e-04 - mae: 0.0124 - val_loss:
4.4588e-04 - val_mae: 0.0143
Epoch 34/50

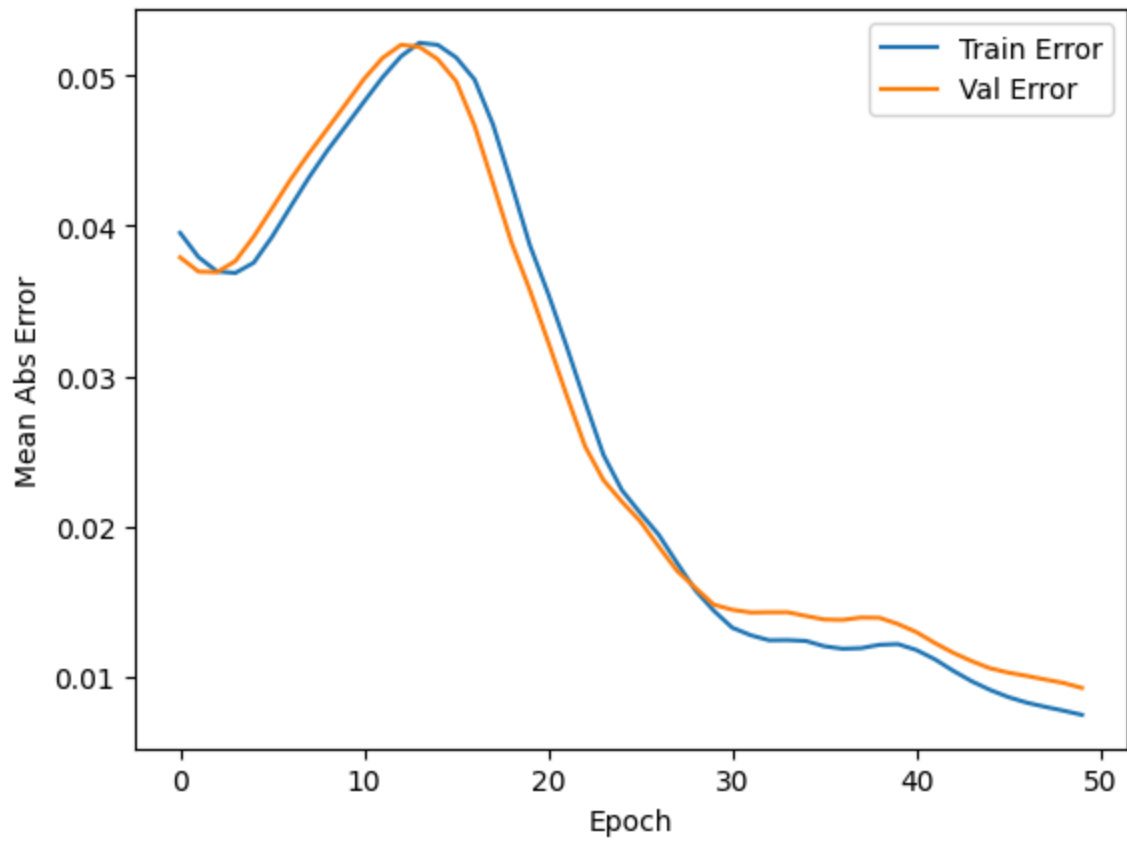
1/1  0s 135ms/step - loss: 2.5315e-04 - mae: 0.0125 - val_loss:
4.4602e-04 - val_mae: 0.0143
Epoch 35/50

1/1  0s 162ms/step - loss: 2.5019e-04 - mae: 0.0124 - val_loss:
4.4112e-04 - val_mae: 0.0141
Epoch 36/50

1/1  0s 130ms/step - loss: 2.4258e-04 - mae: 0.0120 - val_loss:
4.3697e-04 - val_mae: 0.0138
Epoch 37/50

1/1  0s 116ms/step - loss: 2.3607e-04 - mae: 0.0119 - val_loss:
4.3741e-04 - val_mae: 0.0138

Epoch 38/50
1/1  0s 179ms/step - loss: 2.3434e-04 - mae: 0.0119 - val_loss: 4.4065e-04 - val_mae: 0.0140
Epoch 39/50
1/1  0s 104ms/step - loss: 2.3565e-04 - mae: 0.0121 - val_loss: 4.4263e-04 - val_mae: 0.0139
Epoch 40/50
1/1  0s 57ms/step - loss: 2.3591e-04 - mae: 0.0122 - val_loss: 4.4071e-04 - val_mae: 0.0135
Epoch 41/50
1/1  0s 61ms/step - loss: 2.3259e-04 - mae: 0.0118 - val_loss: 4.3403e-04 - val_mae: 0.0130
Epoch 42/50
1/1  0s 71ms/step - loss: 2.2485e-04 - mae: 0.0112 - val_loss: 4.2302e-04 - val_mae: 0.0123
Epoch 43/50
1/1  0s 63ms/step - loss: 2.1332e-04 - mae: 0.0104 - val_loss: 4.0946e-04 - val_mae: 0.0116
Epoch 44/50
1/1  0s 58ms/step - loss: 1.9965e-04 - mae: 0.0097 - val_loss: 3.9462e-04 - val_mae: 0.0111
Epoch 45/50
1/1  0s 64ms/step - loss: 1.8502e-04 - mae: 0.0091 - val_loss: 3.7897e-04 - val_mae: 0.0106
Epoch 46/50
1/1  0s 60ms/step - loss: 1.6985e-04 - mae: 0.0087 - val_loss: 3.6317e-04 - val_mae: 0.0103
Epoch 47/50
1/1  0s 76ms/step - loss: 1.5473e-04 - mae: 0.0083 - val_loss: 3.4826e-04 - val_mae: 0.0101
Epoch 48/50
1/1  0s 59ms/step - loss: 1.4070e-04 - mae: 0.0080 - val_loss: 3.3492e-04 - val_mae: 0.0098
Epoch 49/50
1/1  0s 57ms/step - loss: 1.2841e-04 - mae: 0.0078 - val_loss: 3.2300e-04 - val_mae: 0.0096
Epoch 50/50
1/1  0s 64ms/step - loss: 1.1771e-04 - mae: 0.0075 - val_loss: 3.1196e-04 - val_mae: 0.0093




Entrenando fold 2...


Epoch 1/50

1/1  3s 3s/step - loss: 0.0162 - mae: 0.0358 - val_loss: 0.0155
- val_mae: 0.0346


Epoch 2/50

1/1  0s 123ms/step - loss: 0.0156 - mae: 0.0348 - val_loss: 0.0148 - val_mae: 0.0343


Epoch 3/50

1/1  0s 61ms/step - loss: 0.0149 - mae: 0.0346 - val_loss: 0.0141 - val_mae: 0.0347


Epoch 4/50

1/1  0s 63ms/step - loss: 0.0142 - mae: 0.0350 - val_loss: 0.0132 - val_mae: 0.0353


Epoch 5/50

1/1  0s 61ms/step - loss: 0.0134 - mae: 0.0357 - val_loss: 0.0123 - val_mae: 0.0361


Epoch 6/50

1/1  0s 58ms/step - loss: 0.0124 - mae: 0.0364 - val_loss: 0.0113 - val_mae: 0.0370


Epoch 7/50

1/1  0s 65ms/step - loss: 0.0114 - mae: 0.0373 - val_loss: 0.0102 - val_mae: 0.0381


Epoch 8/50

1/1  0s 60ms/step - loss: 0.0103 - mae: 0.0385 - val_loss: 0.0091 - val_mae: 0.0397


Epoch 9/50

1/1  0s 144ms/step - loss: 0.0092 - mae: 0.0401 - val_loss: 0.0080 - val_mae: 0.0413


Epoch 10/50

1/1  0s 67ms/step - loss: 0.0082 - mae: 0.0417 - val_loss: 0.0071 - val_mae: 0.0425


Epoch 11/50

1/1  0s 57ms/step - loss: 0.0072 - mae: 0.0430 - val_loss: 0.0063 - val_mae: 0.0432


Epoch 12/50

1/1  0s 60ms/step - loss: 0.0064 - mae: 0.0437 - val_loss: 0.0058 - val_mae: 0.0437


Epoch 13/50

1/1  0s 69ms/step - loss: 0.0059 - mae: 0.0443 - val_loss: 0.0054 - val_mae: 0.0440


Epoch 14/50

1/1  0s 56ms/step - loss: 0.0055 - mae: 0.0444 - val_loss: 0.0052 - val_mae: 0.0436


Epoch 15/50

1/1  0s 141ms/step - loss: 0.0053 - mae: 0.0438 - val_loss: 0.0049 - val_mae: 0.0427


Epoch 16/50

1/1  0s 57ms/step - loss: 0.0050 - mae: 0.0427 - val_loss: 0.0046 - val_mae: 0.0413


Epoch 17/50


1/1  0s 66ms/step - loss: 0.0047 - mae: 0.0414 - val_loss: 0.0043 - val_mae: 0.0401


Epoch 18/50


1/1  0s 56ms/step - loss: 0.0043 - mae: 0.0402 - val_loss: 0.0038 - val_mae: 0.0381


Epoch 19/50


1/1  0s 58ms/step - loss: 0.0039 - mae: 0.0382 - val_loss: 0.003
4 - val_mae: 0.0358
Epoch 20/50


1/1  0s 57ms/step - loss: 0.0035 - mae: 0.0358 - val_loss: 0.003
1 - val_mae: 0.0339
Epoch 21/50


1/1  0s 68ms/step - loss: 0.0031 - mae: 0.0339 - val_loss: 0.002
8 - val_mae: 0.0319
Epoch 22/50


1/1  0s 149ms/step - loss: 0.0028 - mae: 0.0320 - val_loss: 0.00
25 - val_mae: 0.0299
Epoch 23/50

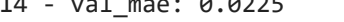
1/1  0s 67ms/step - loss: 0.0026 - mae: 0.0299 - val_loss: 0.002
3 - val_mae: 0.0285
Epoch 24/50


1/1  0s 61ms/step - loss: 0.0024 - mae: 0.0285 - val_loss: 0.002
2 - val_mae: 0.0270
Epoch 25/50


1/1  0s 66ms/step - loss: 0.0022 - mae: 0.0269 - val_loss: 0.002
0 - val_mae: 0.0252
Epoch 26/50


1/1  0s 61ms/step - loss: 0.0020 - mae: 0.0251 - val_loss: 0.001
8 - val_mae: 0.0240
Epoch 27/50


1/1  0s 62ms/step - loss: 0.0018 - mae: 0.0240 - val_loss: 0.001
6 - val_mae: 0.0232
Epoch 28/50


1/1  0s 120ms/step - loss: 0.0016 - mae: 0.0233 - val_loss: 0.00
14 - val_mae: 0.0225
Epoch 29/50


1/1  0s 63ms/step - loss: 0.0014 - mae: 0.0226 - val_loss: 0.001
2 - val_mae: 0.0215
Epoch 30/50


1/1  0s 104ms/step - loss: 0.0013 - mae: 0.0218 - val_loss: 0.00
11 - val_mae: 0.0203
Epoch 31/50


1/1  0s 133ms/step - loss: 0.0011 - mae: 0.0207 - val_loss: 9.42
71e-04 - val_mae: 0.0189
Epoch 32/50


1/1  0s 118ms/step - loss: 9.8778e-04 - mae: 0.0195 - val_loss:
8.4319e-04 - val_mae: 0.0181
Epoch 33/50














1/1  0s 120ms/step - loss: 8.8757e-04 - mae: 0.0186 - val_loss:
7.5770e-04 - val_mae: 0.0175
Epoch 34/50

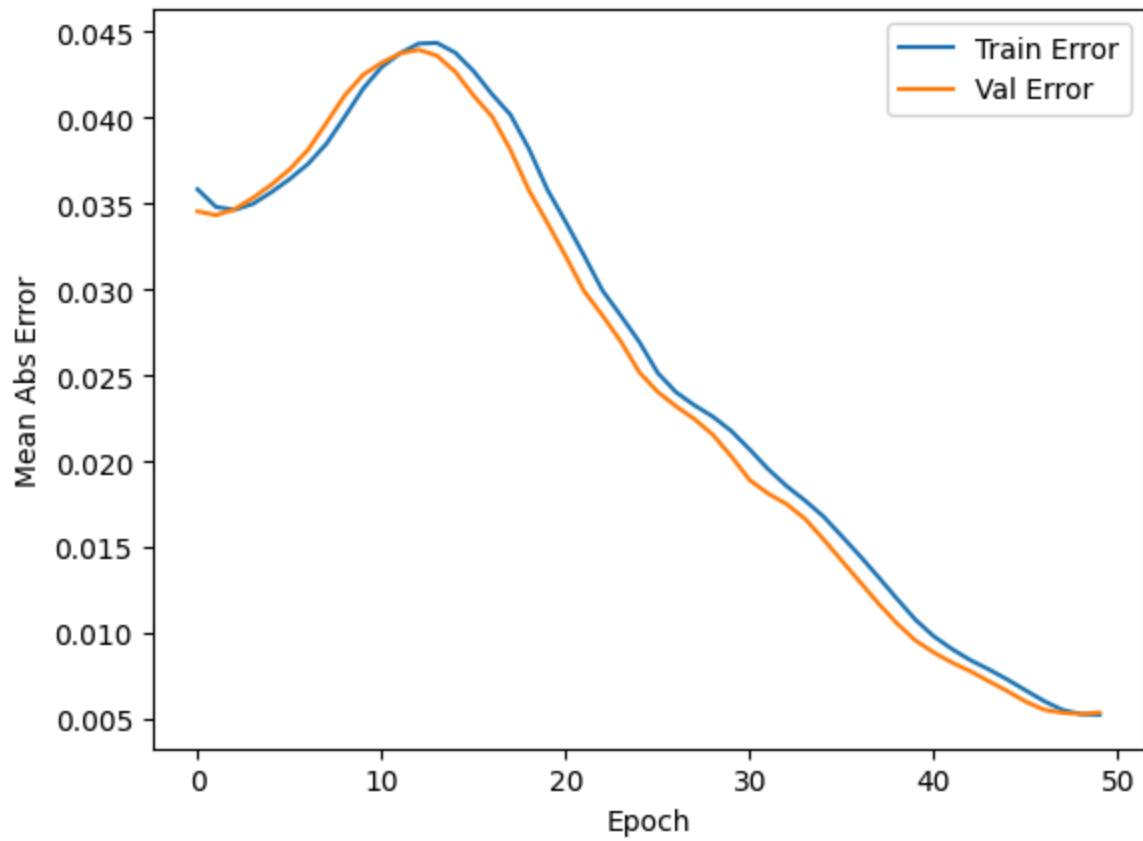
1/1  0s 115ms/step - loss: 8.0146e-04 - mae: 0.0177 - val_loss:
6.7738e-04 - val_mae: 0.0167
Epoch 35/50

1/1  0s 159ms/step - loss: 7.2045e-04 - mae: 0.0168 - val_loss:
5.9404e-04 - val_mae: 0.0155
Epoch 36/50

1/1  0s 119ms/step - loss: 6.3637e-04 - mae: 0.0156 - val_loss:
5.0585e-04 - val_mae: 0.0142
Epoch 37/50


1/1  0s 87ms/step - loss: 5.4741e-04 - mae: 0.0145 - val_loss:
4.1796e-04 - val_mae: 0.0130

Epoch 38/50
1/1  0s 78ms/step - loss: 4.5874e-04 - mae: 0.0132 - val_loss: 3.3839e-04 - val_mae: 0.0117
Epoch 39/50
1/1  0s 164ms/step - loss: 3.7833e-04 - mae: 0.0120 - val_loss: 2.7347e-04 - val_mae: 0.0106
Epoch 40/50
1/1  0s 137ms/step - loss: 3.1254e-04 - mae: 0.0108 - val_loss: 2.2539e-04 - val_mae: 0.0096
Epoch 41/50
1/1  0s 129ms/step - loss: 2.6352e-04 - mae: 0.0098 - val_loss: 1.9197e-04 - val_mae: 0.0089
Epoch 42/50
1/1  0s 117ms/step - loss: 2.2915e-04 - mae: 0.0091 - val_loss: 1.6852e-04 - val_mae: 0.0083
Epoch 43/50
1/1  0s 107ms/step - loss: 2.0472e-04 - mae: 0.0084 - val_loss: 1.5000e-04 - val_mae: 0.0078
Epoch 44/50
1/1  0s 139ms/step - loss: 1.8527e-04 - mae: 0.0079 - val_loss: 1.3294e-04 - val_mae: 0.0072
Epoch 45/50
1/1  0s 83ms/step - loss: 1.6737e-04 - mae: 0.0073 - val_loss: 1.1621e-04 - val_mae: 0.0066
Epoch 46/50
1/1  0s 134ms/step - loss: 1.4991e-04 - mae: 0.0067 - val_loss: 1.0072e-04 - val_mae: 0.0060
Epoch 47/50
1/1  0s 106ms/step - loss: 1.3387e-04 - mae: 0.0060 - val_loss: 8.8462e-05 - val_mae: 0.0055
Epoch 48/50
1/1  0s 129ms/step - loss: 1.2129e-04 - mae: 0.0055 - val_loss: 8.1165e-05 - val_mae: 0.0054
Epoch 49/50
1/1  0s 127ms/step - loss: 1.1392e-04 - mae: 0.0053 - val_loss: 7.9277e-05 - val_mae: 0.0053
Epoch 50/50
1/1  0s 158ms/step - loss: 1.1222e-04 - mae: 0.0052 - val_loss: 8.1675e-05 - val_mae: 0.0054




Entrenando fold 3...


Epoch 1/50

2/2  2s 225ms/step - loss: 0.0178 - mae: 0.0348 - val_loss: 0.0158 - val_mae: 0.0331


Epoch 2/50

2/2  0s 34ms/step - loss: 0.0164 - mae: 0.0333 - val_loss: 0.0144 - val_mae: 0.0331


Epoch 3/50

2/2  0s 46ms/step - loss: 0.0150 - mae: 0.0335 - val_loss: 0.0131 - val_mae: 0.0356


Epoch 4/50

2/2  0s 34ms/step - loss: 0.0136 - mae: 0.0360 - val_loss: 0.0115 - val_mae: 0.0392


Epoch 5/50

2/2  0s 43ms/step - loss: 0.0122 - mae: 0.0396 - val_loss: 0.0099 - val_mae: 0.0428


Epoch 6/50

2/2  0s 34ms/step - loss: 0.0104 - mae: 0.0432 - val_loss: 0.0083 - val_mae: 0.0459


Epoch 7/50

2/2  0s 51ms/step - loss: 0.0088 - mae: 0.0464 - val_loss: 0.0069 - val_mae: 0.0476


Epoch 8/50

2/2  0s 38ms/step - loss: 0.0072 - mae: 0.0480 - val_loss: 0.0056 - val_mae: 0.0477


Epoch 9/50

2/2  0s 45ms/step - loss: 0.0060 - mae: 0.0485 - val_loss: 0.0046 - val_mae: 0.0454


Epoch 10/50

2/2  0s 58ms/step - loss: 0.0048 - mae: 0.0461 - val_loss: 0.0036 - val_mae: 0.0431


Epoch 11/50

2/2  0s 40ms/step - loss: 0.0038 - mae: 0.0437 - val_loss: 0.0029 - val_mae: 0.0406


Epoch 12/50

2/2  0s 41ms/step - loss: 0.0031 - mae: 0.0411 - val_loss: 0.0023 - val_mae: 0.0368


Epoch 13/50

2/2  0s 38ms/step - loss: 0.0025 - mae: 0.0372 - val_loss: 0.0018 - val_mae: 0.0324


Epoch 14/50

2/2  0s 36ms/step - loss: 0.0019 - mae: 0.0329 - val_loss: 0.0013 - val_mae: 0.0281


Epoch 15/50

2/2  0s 35ms/step - loss: 0.0014 - mae: 0.0285 - val_loss: 8.9244e-04 - val_mae: 0.0232


Epoch 16/50

2/2  0s 35ms/step - loss: 9.7749e-04 - mae: 0.0237 - val_loss: 5.5556e-04 - val_mae: 0.0173


Epoch 17/50


2/2  0s 36ms/step - loss: 6.4493e-04 - mae: 0.0180 - val_loss: 3.5651e-04 - val_mae: 0.0128


Epoch 18/50


2/2  0s 66ms/step - loss: 4.5794e-04 - mae: 0.0138 - val_loss: 2.7068e-04 - val_mae: 0.0118


Epoch 19/50


2/2  0s 44ms/step - loss: 3.7776e-04 - mae: 0.0130 - val_loss: 2.0933e-04 - val_mae: 0.0105
Epoch 20/50


2/2  0s 44ms/step - loss: 3.0725e-04 - mae: 0.0117 - val_loss: 1.6006e-04 - val_mae: 0.0098
Epoch 21/50


2/2  0s 35ms/step - loss: 2.3905e-04 - mae: 0.0109 - val_loss: 1.5672e-04 - val_mae: 0.0094
Epoch 22/50


2/2  0s 38ms/step - loss: 2.0933e-04 - mae: 0.0100 - val_loss: 1.8708e-04 - val_mae: 0.0094
Epoch 23/50


2/2  0s 34ms/step - loss: 2.1804e-04 - mae: 0.0097 - val_loss: 2.1724e-04 - val_mae: 0.0089
Epoch 24/50


2/2  0s 42ms/step - loss: 2.2563e-04 - mae: 0.0089 - val_loss: 1.9634e-04 - val_mae: 0.0089
Epoch 25/50


2/2  0s 39ms/step - loss: 2.1966e-04 - mae: 0.0090 - val_loss: 1.6599e-04 - val_mae: 0.0088
Epoch 26/50


2/2  0s 39ms/step - loss: 2.0612e-04 - mae: 0.0092 - val_loss: 1.4259e-04 - val_mae: 0.0084
Epoch 27/50


2/2  0s 38ms/step - loss: 1.9006e-04 - mae: 0.0088 - val_loss: 1.2833e-04 - val_mae: 0.0078
Epoch 28/50


2/2  0s 34ms/step - loss: 1.7671e-04 - mae: 0.0083 - val_loss: 1.2083e-04 - val_mae: 0.0076
Epoch 29/50


2/2  0s 50ms/step - loss: 1.7469e-04 - mae: 0.0084 - val_loss: 1.0256e-04 - val_mae: 0.0070
Epoch 30/50


2/2  0s 65ms/step - loss: 1.6334e-04 - mae: 0.0080 - val_loss: 8.3250e-05 - val_mae: 0.0059
Epoch 31/50


2/2  0s 43ms/step - loss: 1.4602e-04 - mae: 0.0069 - val_loss: 7.0586e-05 - val_mae: 0.0049
Epoch 32/50


2/2  0s 36ms/step - loss: 1.3888e-04 - mae: 0.0060 - val_loss: 6.4648e-05 - val_mae: 0.0047
Epoch 33/50














2/2  0s 41ms/step - loss: 1.3400e-04 - mae: 0.0058 - val_loss: 5.8453e-05 - val_mae: 0.0047
Epoch 34/50

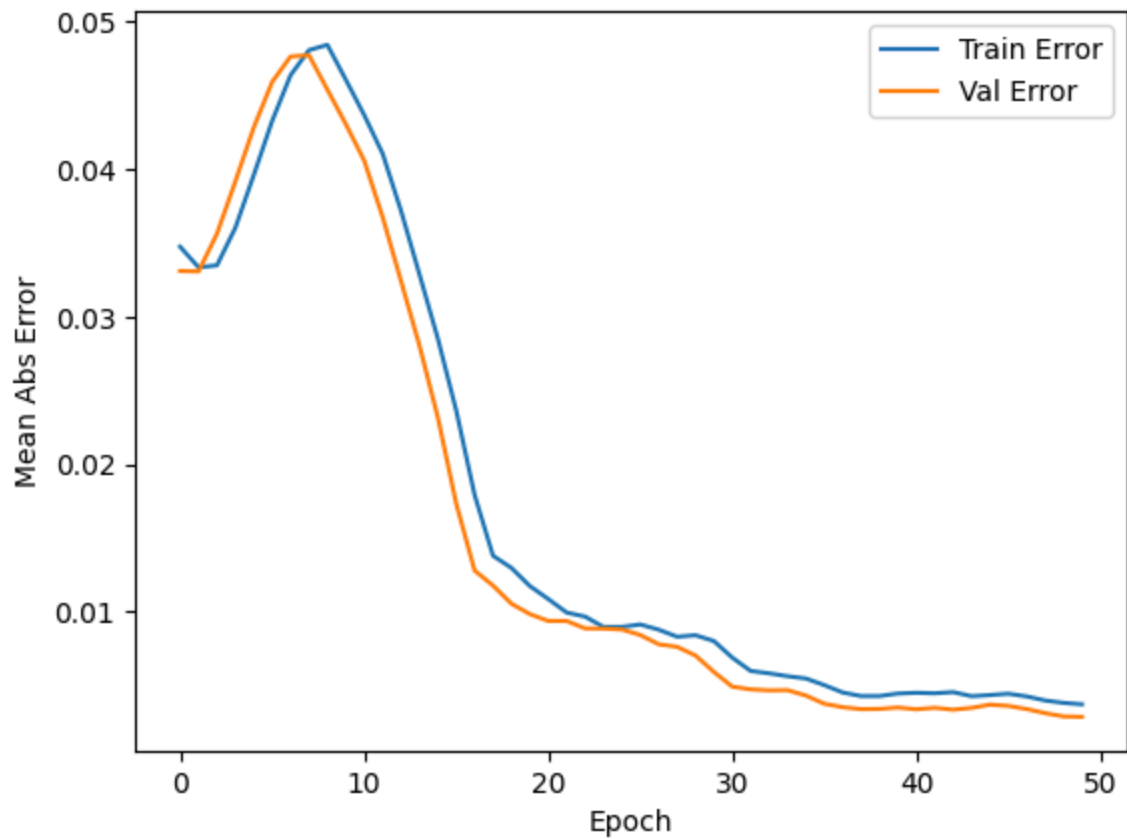
2/2  0s 36ms/step - loss: 1.2017e-04 - mae: 0.0056 - val_loss: 6.6949e-05 - val_mae: 0.0047
Epoch 35/50

2/2  0s 40ms/step - loss: 1.1448e-04 - mae: 0.0054 - val_loss: 6.4815e-05 - val_mae: 0.0043
Epoch 36/50

2/2  0s 37ms/step - loss: 1.1453e-04 - mae: 0.0050 - val_loss: 5.0464e-05 - val_mae: 0.0038
Epoch 37/50


2/2  0s 36ms/step - loss: 1.0759e-04 - mae: 0.0045 - val_loss: 4.7183e-05 - val_mae: 0.0035

Epoch 38/50
2/2  0s 37ms/step - loss: 1.0443e-04 - mae: 0.0043 - val_loss: 4.4038e-05 - val_mae: 0.0034
Epoch 39/50
2/2  0s 34ms/step - loss: 1.0314e-04 - mae: 0.0043 - val_loss: 4.5221e-05 - val_mae: 0.0034
Epoch 40/50
2/2  0s 58ms/step - loss: 1.2221e-04 - mae: 0.0045 - val_loss: 5.3502e-05 - val_mae: 0.0035
Epoch 41/50
2/2  0s 42ms/step - loss: 1.3777e-04 - mae: 0.0045 - val_loss: 4.5974e-05 - val_mae: 0.0034
Epoch 42/50
2/2  0s 36ms/step - loss: 1.2428e-04 - mae: 0.0045 - val_loss: 4.5951e-05 - val_mae: 0.0035
Epoch 43/50
2/2  0s 35ms/step - loss: 1.0972e-04 - mae: 0.0046 - val_loss: 4.7874e-05 - val_mae: 0.0034
Epoch 44/50
2/2  0s 35ms/step - loss: 1.0487e-04 - mae: 0.0043 - val_loss: 5.1773e-05 - val_mae: 0.0035
Epoch 45/50
2/2  0s 38ms/step - loss: 1.0569e-04 - mae: 0.0044 - val_loss: 5.6147e-05 - val_mae: 0.0037
Epoch 46/50
2/2  0s 39ms/step - loss: 1.0695e-04 - mae: 0.0045 - val_loss: 5.3561e-05 - val_mae: 0.0036
Epoch 47/50
2/2  0s 35ms/step - loss: 1.0482e-04 - mae: 0.0043 - val_loss: 5.1065e-05 - val_mae: 0.0034
Epoch 48/50
2/2  0s 41ms/step - loss: 1.0284e-04 - mae: 0.0040 - val_loss: 4.8101e-05 - val_mae: 0.0031
Epoch 49/50
2/2  0s 42ms/step - loss: 1.0195e-04 - mae: 0.0038 - val_loss: 4.6162e-05 - val_mae: 0.0029
Epoch 50/50
2/2  0s 38ms/step - loss: 1.0111e-04 - mae: 0.0037 - val_loss: 4.9035e-05 - val_mae: 0.0029




Entrenando fold 4...


Epoch 1/50

2/2  2s 225ms/step - loss: 0.0166 - mae: 0.0328 - val_loss: 0.0160 - val_mae: 0.0320


Epoch 2/50

2/2  0s 36ms/step - loss: 0.0158 - mae: 0.0316 - val_loss: 0.0152 - val_mae: 0.0322


Epoch 3/50

2/2  0s 35ms/step - loss: 0.0150 - mae: 0.0316 - val_loss: 0.0143 - val_mae: 0.0335


Epoch 4/50

2/2  0s 37ms/step - loss: 0.0141 - mae: 0.0328 - val_loss: 0.0131 - val_mae: 0.0357


Epoch 5/50

2/2  0s 36ms/step - loss: 0.0130 - mae: 0.0349 - val_loss: 0.0118 - val_mae: 0.0387


Epoch 6/50

2/2  0s 38ms/step - loss: 0.0116 - mae: 0.0379 - val_loss: 0.0103 - val_mae: 0.0420


Epoch 7/50

2/2  0s 37ms/step - loss: 0.0102 - mae: 0.0412 - val_loss: 0.0089 - val_mae: 0.0451


Epoch 8/50

2/2  0s 34ms/step - loss: 0.0088 - mae: 0.0444 - val_loss: 0.0075 - val_mae: 0.0475


Epoch 9/50

2/2  0s 45ms/step - loss: 0.0074 - mae: 0.0467 - val_loss: 0.0064 - val_mae: 0.0479


Epoch 10/50

2/2  0s 40ms/step - loss: 0.0063 - mae: 0.0472 - val_loss: 0.0052 - val_mae: 0.0461


Epoch 11/50

2/2  0s 59ms/step - loss: 0.0052 - mae: 0.0454 - val_loss: 0.0041 - val_mae: 0.0432


Epoch 12/50

2/2  0s 46ms/step - loss: 0.0041 - mae: 0.0426 - val_loss: 0.0030 - val_mae: 0.0393


Epoch 13/50

2/2  0s 35ms/step - loss: 0.0030 - mae: 0.0386 - val_loss: 0.0022 - val_mae: 0.0329


Epoch 14/50

2/2  0s 35ms/step - loss: 0.0022 - mae: 0.0323 - val_loss: 0.0014 - val_mae: 0.0269


Epoch 15/50

2/2  0s 39ms/step - loss: 0.0015 - mae: 0.0265 - val_loss: 9.0058e-04 - val_mae: 0.0209


Epoch 16/50

2/2  0s 37ms/step - loss: 9.4146e-04 - mae: 0.0208 - val_loss: 5.7574e-04 - val_mae: 0.0155


Epoch 17/50


2/2  0s 38ms/step - loss: 6.3279e-04 - mae: 0.0160 - val_loss: 3.9306e-04 - val_mae: 0.0136


Epoch 18/50


2/2  0s 62ms/step - loss: 4.6019e-04 - mae: 0.0147 - val_loss: 2.8004e-04 - val_mae: 0.0118


Epoch 19/50


2/2  0s 43ms/step - loss: 3.4850e-04 - mae: 0.0128 - val_loss: 2.2225e-04 - val_mae: 0.0103
Epoch 20/50


2/2  0s 35ms/step - loss: 2.8811e-04 - mae: 0.0111 - val_loss: 2.2015e-04 - val_mae: 0.0114
Epoch 21/50


2/2  0s 41ms/step - loss: 2.7955e-04 - mae: 0.0121 - val_loss: 2.2815e-04 - val_mae: 0.0117
Epoch 22/50


2/2  0s 36ms/step - loss: 2.8309e-04 - mae: 0.0124 - val_loss: 2.2197e-04 - val_mae: 0.0113
Epoch 23/50


2/2  0s 35ms/step - loss: 2.7338e-04 - mae: 0.0119 - val_loss: 2.0672e-04 - val_mae: 0.0105
Epoch 24/50


2/2  0s 65ms/step - loss: 2.5076e-04 - mae: 0.0109 - val_loss: 1.9731e-04 - val_mae: 0.0103
Epoch 25/50


2/2  0s 71ms/step - loss: 2.4320e-04 - mae: 0.0106 - val_loss: 1.9130e-04 - val_mae: 0.0104
Epoch 26/50


2/2  0s 44ms/step - loss: 2.3357e-04 - mae: 0.0104 - val_loss: 1.7424e-04 - val_mae: 0.0095
Epoch 27/50


2/2  0s 68ms/step - loss: 2.0989e-04 - mae: 0.0093 - val_loss: 1.6901e-04 - val_mae: 0.0085
Epoch 28/50


2/2  0s 77ms/step - loss: 2.0575e-04 - mae: 0.0082 - val_loss: 1.5237e-04 - val_mae: 0.0073
Epoch 29/50


2/2  0s 67ms/step - loss: 1.9043e-04 - mae: 0.0071 - val_loss: 1.0651e-04 - val_mae: 0.0061
Epoch 30/50


2/2  0s 51ms/step - loss: 1.4756e-04 - mae: 0.0060 - val_loss: 8.5262e-05 - val_mae: 0.0059
Epoch 31/50


2/2  0s 47ms/step - loss: 1.3352e-04 - mae: 0.0062 - val_loss: 8.5922e-05 - val_mae: 0.0057
Epoch 32/50


2/2  0s 51ms/step - loss: 1.3726e-04 - mae: 0.0060 - val_loss: 7.5345e-05 - val_mae: 0.0049
Epoch 33/50














2/2  0s 48ms/step - loss: 1.2852e-04 - mae: 0.0054 - val_loss: 6.2317e-05 - val_mae: 0.0043
Epoch 34/50

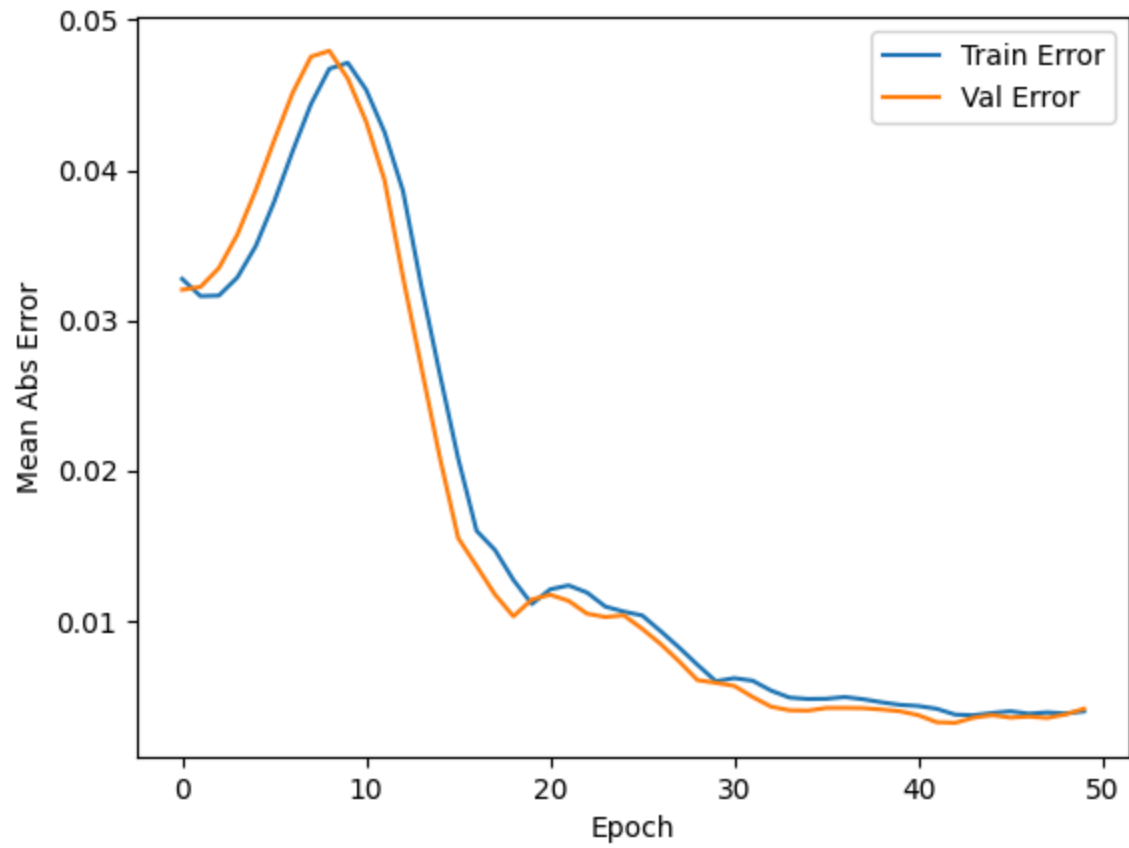
2/2  0s 70ms/step - loss: 1.1695e-04 - mae: 0.0049 - val_loss: 5.6756e-05 - val_mae: 0.0041
Epoch 35/50

2/2  0s 82ms/step - loss: 1.1084e-04 - mae: 0.0048 - val_loss: 5.7580e-05 - val_mae: 0.0040
Epoch 36/50

2/2  0s 66ms/step - loss: 1.1029e-04 - mae: 0.0048 - val_loss: 6.1153e-05 - val_mae: 0.0042
Epoch 37/50


2/2  0s 45ms/step - loss: 1.1354e-04 - mae: 0.0049 - val_loss: 6.4763e-05 - val_mae: 0.0042

Epoch 38/50
2/2  0s 49ms/step - loss: 1.1473e-04 - mae: 0.0048 - val_loss: 6.5607e-05 - val_mae: 0.0042
Epoch 39/50
2/2  0s 65ms/step - loss: 1.1275e-04 - mae: 0.0046 - val_loss: 6.3489e-05 - val_mae: 0.0041
Epoch 40/50
2/2  0s 50ms/step - loss: 1.0547e-04 - mae: 0.0044 - val_loss: 6.3619e-05 - val_mae: 0.0040
Epoch 41/50
2/2  0s 65ms/step - loss: 1.0724e-04 - mae: 0.0043 - val_loss: 6.0619e-05 - val_mae: 0.0037
Epoch 42/50
2/2  0s 57ms/step - loss: 1.1051e-04 - mae: 0.0042 - val_loss: 5.0640e-05 - val_mae: 0.0033
Epoch 43/50
2/2  0s 51ms/step - loss: 1.0118e-04 - mae: 0.0038 - val_loss: 4.9382e-05 - val_mae: 0.0032
Epoch 44/50
2/2  0s 74ms/step - loss: 9.9072e-05 - mae: 0.0037 - val_loss: 5.3033e-05 - val_mae: 0.0036
Epoch 45/50
2/2  0s 49ms/step - loss: 9.7473e-05 - mae: 0.0038 - val_loss: 5.5514e-05 - val_mae: 0.0037
Epoch 46/50
2/2  0s 58ms/step - loss: 1.0182e-04 - mae: 0.0040 - val_loss: 5.6752e-05 - val_mae: 0.0036
Epoch 47/50
2/2  0s 75ms/step - loss: 1.0287e-04 - mae: 0.0038 - val_loss: 6.0197e-05 - val_mae: 0.0036
Epoch 48/50
2/2  0s 46ms/step - loss: 1.0708e-04 - mae: 0.0039 - val_loss: 5.9139e-05 - val_mae: 0.0035
Epoch 49/50
2/2  0s 50ms/step - loss: 1.0357e-04 - mae: 0.0038 - val_loss: 5.9144e-05 - val_mae: 0.0038
Epoch 50/50
2/2  0s 56ms/step - loss: 1.0325e-04 - mae: 0.0040 - val_loss: 7.1999e-05 - val_mae: 0.0042




Entrenando fold 5...


Epoch 1/50

2/2  2s 228ms/step - loss: 0.0171 - mae: 0.0343 - val_loss: 0.0162 - val_mae: 0.0326


Epoch 2/50

2/2  0s 43ms/step - loss: 0.0157 - mae: 0.0326 - val_loss: 0.0146 - val_mae: 0.0336


Epoch 3/50

2/2  0s 34ms/step - loss: 0.0142 - mae: 0.0335 - val_loss: 0.0131 - val_mae: 0.0355


Epoch 4/50

2/2  0s 40ms/step - loss: 0.0127 - mae: 0.0353 - val_loss: 0.0116 - val_mae: 0.0376


Epoch 5/50

2/2  0s 38ms/step - loss: 0.0113 - mae: 0.0374 - val_loss: 0.0100 - val_mae: 0.0407


Epoch 6/50

2/2  0s 35ms/step - loss: 0.0097 - mae: 0.0405 - val_loss: 0.0084 - val_mae: 0.0446


Epoch 7/50

2/2  0s 38ms/step - loss: 0.0081 - mae: 0.0444 - val_loss: 0.0070 - val_mae: 0.0483


Epoch 8/50

2/2  0s 43ms/step - loss: 0.0068 - mae: 0.0481 - val_loss: 0.0059 - val_mae: 0.0502


Epoch 9/50

2/2  0s 64ms/step - loss: 0.0057 - mae: 0.0498 - val_loss: 0.0050 - val_mae: 0.0485


Epoch 10/50

2/2  0s 34ms/step - loss: 0.0049 - mae: 0.0481 - val_loss: 0.0042 - val_mae: 0.0445


Epoch 11/50

2/2  0s 35ms/step - loss: 0.0041 - mae: 0.0442 - val_loss: 0.0034 - val_mae: 0.0393


Epoch 12/50

2/2  0s 35ms/step - loss: 0.0033 - mae: 0.0389 - val_loss: 0.0026 - val_mae: 0.0350


Epoch 13/50

2/2  0s 40ms/step - loss: 0.0026 - mae: 0.0347 - val_loss: 0.0020 - val_mae: 0.0304


Epoch 14/50

2/2  0s 61ms/step - loss: 0.0020 - mae: 0.0302 - val_loss: 0.0015 - val_mae: 0.0261


Epoch 15/50

2/2  0s 38ms/step - loss: 0.0015 - mae: 0.0260 - val_loss: 0.0010 - val_mae: 0.0221


Epoch 16/50

2/2  0s 35ms/step - loss: 0.0010 - mae: 0.0222 - val_loss: 7.2120e-04 - val_mae: 0.0183


Epoch 17/50


2/2  0s 36ms/step - loss: 7.4271e-04 - mae: 0.0185 - val_loss: 5.0101e-04 - val_mae: 0.0145


Epoch 18/50


2/2  0s 61ms/step - loss: 5.2655e-04 - mae: 0.0149 - val_loss: 3.3950e-04 - val_mae: 0.0125


Epoch 19/50


2/2  0s 35ms/step - loss: 3.8155e-04 - mae: 0.0131 - val_loss: 2.3826e-04 - val_mae: 0.0114
Epoch 20/50


2/2  0s 36ms/step - loss: 2.9711e-04 - mae: 0.0123 - val_loss: 1.8615e-04 - val_mae: 0.0107
Epoch 21/50


2/2  0s 40ms/step - loss: 2.5447e-04 - mae: 0.0115 - val_loss: 1.5554e-04 - val_mae: 0.0098
Epoch 22/50


2/2  0s 42ms/step - loss: 2.3012e-04 - mae: 0.0103 - val_loss: 1.4797e-04 - val_mae: 0.0086
Epoch 23/50


2/2  0s 36ms/step - loss: 2.2866e-04 - mae: 0.0092 - val_loss: 1.5618e-04 - val_mae: 0.0092
Epoch 24/50


2/2  0s 38ms/step - loss: 2.3924e-04 - mae: 0.0097 - val_loss: 1.5443e-04 - val_mae: 0.0096
Epoch 25/50


2/2  0s 36ms/step - loss: 2.3451e-04 - mae: 0.0103 - val_loss: 1.4453e-04 - val_mae: 0.0091
Epoch 26/50


2/2  0s 59ms/step - loss: 2.1507e-04 - mae: 0.0099 - val_loss: 1.2701e-04 - val_mae: 0.0086
Epoch 27/50


2/2  0s 36ms/step - loss: 1.9377e-04 - mae: 0.0095 - val_loss: 1.0228e-04 - val_mae: 0.0077
Epoch 28/50


2/2  0s 61ms/step - loss: 1.8747e-04 - mae: 0.0086 - val_loss: 9.9957e-05 - val_mae: 0.0072
Epoch 29/50


2/2  0s 46ms/step - loss: 1.9389e-04 - mae: 0.0079 - val_loss: 7.8894e-05 - val_mae: 0.0063
Epoch 30/50


2/2  0s 37ms/step - loss: 1.7023e-04 - mae: 0.0072 - val_loss: 6.0551e-05 - val_mae: 0.0054
Epoch 31/50


2/2  0s 37ms/step - loss: 1.3544e-04 - mae: 0.0062 - val_loss: 6.0846e-05 - val_mae: 0.0048
Epoch 32/50


2/2  0s 59ms/step - loss: 1.2578e-04 - mae: 0.0057 - val_loss: 7.4279e-05 - val_mae: 0.0044
Epoch 33/50














2/2  0s 35ms/step - loss: 1.2940e-04 - mae: 0.0053 - val_loss: 6.8804e-05 - val_mae: 0.0043
Epoch 34/50

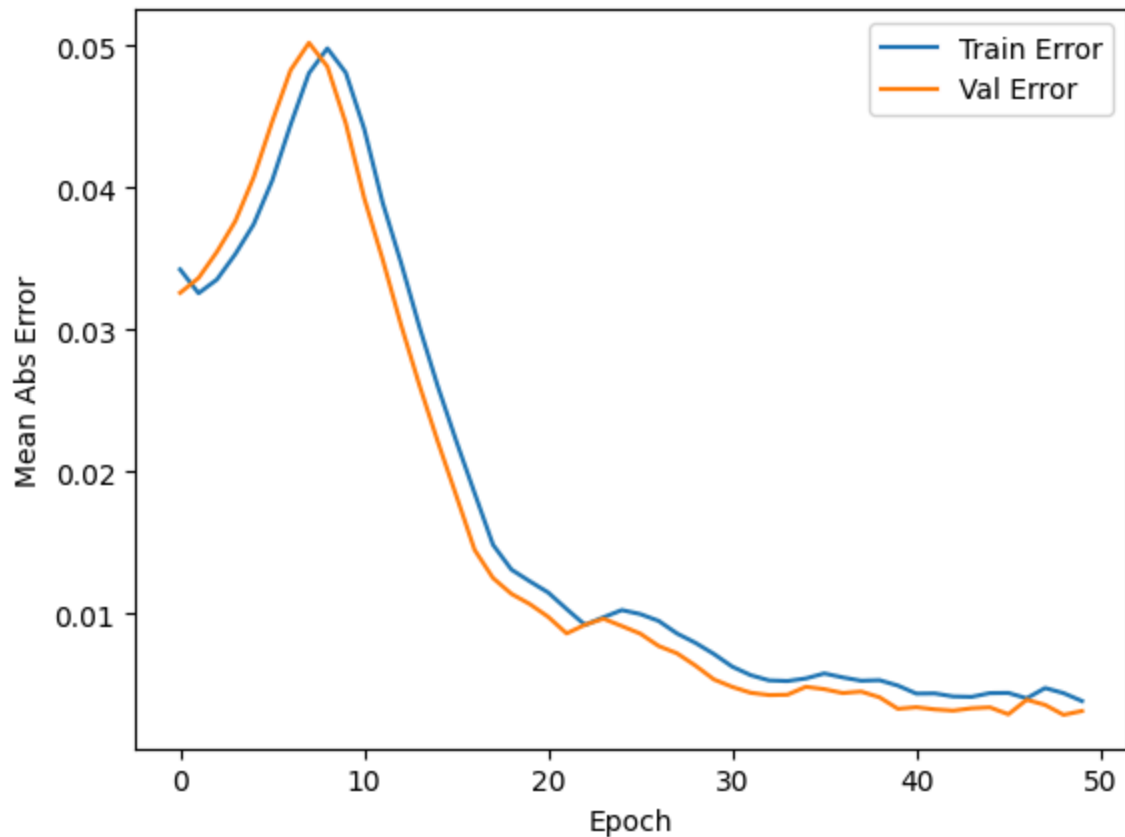
2/2  0s 44ms/step - loss: 1.1778e-04 - mae: 0.0052 - val_loss: 5.1327e-05 - val_mae: 0.0043
Epoch 35/50

2/2  0s 39ms/step - loss: 1.2111e-04 - mae: 0.0055 - val_loss: 5.7226e-05 - val_mae: 0.0048
Epoch 36/50

2/2  0s 34ms/step - loss: 1.4108e-04 - mae: 0.0058 - val_loss: 6.1304e-05 - val_mae: 0.0047
Epoch 37/50

2/2  0s 41ms/step - loss: 1.4933e-04 - mae: 0.0055 - val_loss: 5.5526e-05 - val_mae: 0.0044

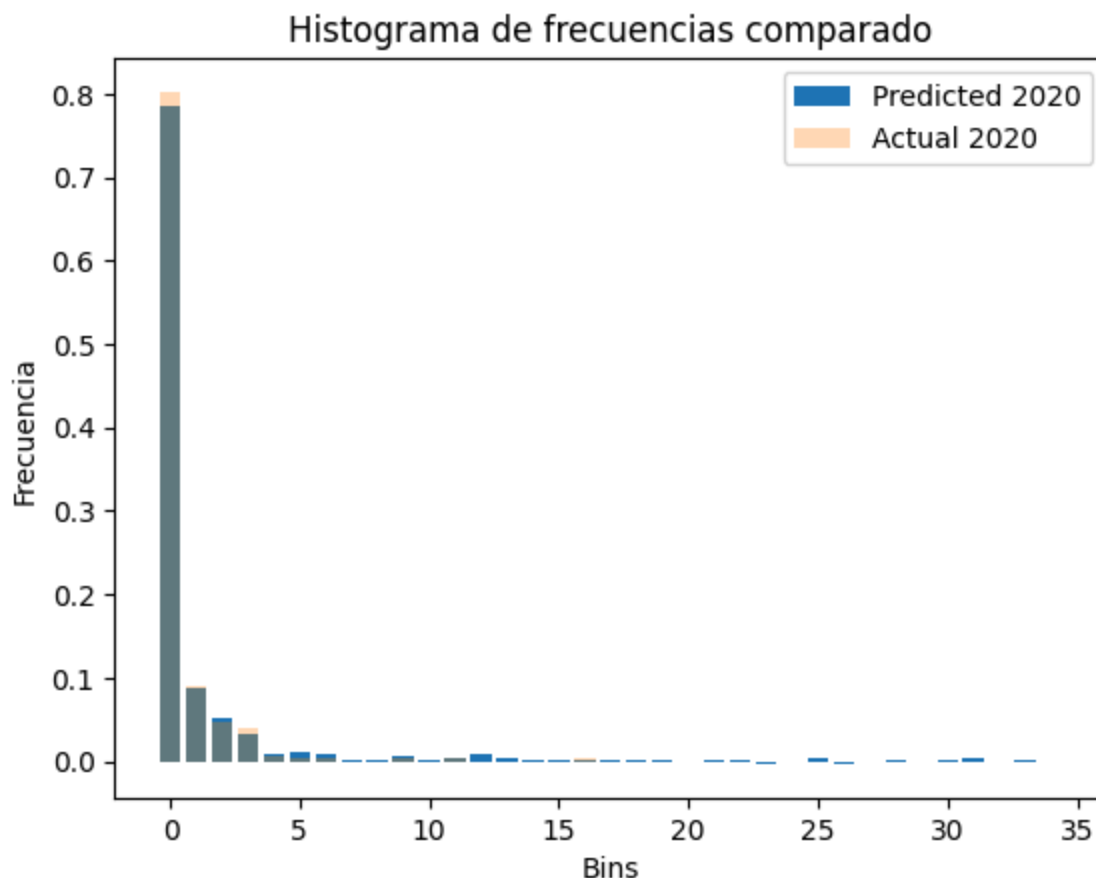
Epoch 38/50
2/2  0s 44ms/step - loss: 1.3898e-04 - mae: 0.0052 - val_loss: 5.2019e-05 - val_mae: 0.0045
Epoch 39/50
2/2  0s 47ms/step - loss: 1.3736e-04 - mae: 0.0053 - val_loss: 4.9523e-05 - val_mae: 0.0041
Epoch 40/50
2/2  0s 38ms/step - loss: 1.3139e-04 - mae: 0.0049 - val_loss: 3.6756e-05 - val_mae: 0.0033
Epoch 41/50
2/2  0s 35ms/step - loss: 1.1482e-04 - mae: 0.0043 - val_loss: 4.1194e-05 - val_mae: 0.0034
Epoch 42/50
2/2  0s 39ms/step - loss: 1.0851e-04 - mae: 0.0044 - val_loss: 4.4378e-05 - val_mae: 0.0033
Epoch 43/50
2/2  0s 38ms/step - loss: 1.0281e-04 - mae: 0.0041 - val_loss: 3.7157e-05 - val_mae: 0.0032
Epoch 44/50
2/2  0s 36ms/step - loss: 1.0543e-04 - mae: 0.0042 - val_loss: 3.6074e-05 - val_mae: 0.0033
Epoch 45/50
2/2  0s 36ms/step - loss: 1.1854e-04 - mae: 0.0044 - val_loss: 4.2969e-05 - val_mae: 0.0034
Epoch 46/50
2/2  0s 39ms/step - loss: 1.2167e-04 - mae: 0.0044 - val_loss: 3.3113e-05 - val_mae: 0.0029
Epoch 47/50
2/2  0s 36ms/step - loss: 1.0523e-04 - mae: 0.0040 - val_loss: 5.1907e-05 - val_mae: 0.0039
Epoch 48/50
2/2  0s 39ms/step - loss: 1.0570e-04 - mae: 0.0047 - val_loss: 4.2200e-05 - val_mae: 0.0036
Epoch 49/50
2/2  0s 44ms/step - loss: 1.0636e-04 - mae: 0.0044 - val_loss: 3.2196e-05 - val_mae: 0.0029
Epoch 50/50
2/2  0s 44ms/step - loss: 1.0682e-04 - mae: 0.0039 - val_loss: 3.4916e-05 - val_mae: 0.0031



Esta línea de código ejecuta la función `predecir_y_graficar`, que realiza una predicción utilizando el modelo entrenado. La función grafica el histograma predicho para el año 2020 y lo compara con el histograma real. El parámetro `modo` especifica si la predicción y graficación se realizan en el modo `'normal'` o `'matricial'`.

```
In [ ]: # Predecir y graficar
predecir_y_graficar(model, X_val, histograms, modo=modo)
```

1/1 ————— 0s 78ms/step



Estas líneas de código realizan la evaluación final del modelo utilizando los datos de validación. La función `model.evaluate` calcula la pérdida (en este caso, el error cuadrático medio) y el error absoluto medio (MAE) en el conjunto de validación. Finalmente, se imprime el valor de MAE, que indica la precisión del modelo en la predicción de los histogramas del año 2020.


















```
In [ ]: # Evaluación final
val_loss, val_mae = model.evaluate(X_val, Y_val)
print(f"Final MAE (Validation): {val_mae}")
```




















1/1 ————— 0s 25ms/step - loss: 1.0028e-04 - mae: 0.0040
 Final MAE (Validation): 0.0040497248992323875















Modo Normal

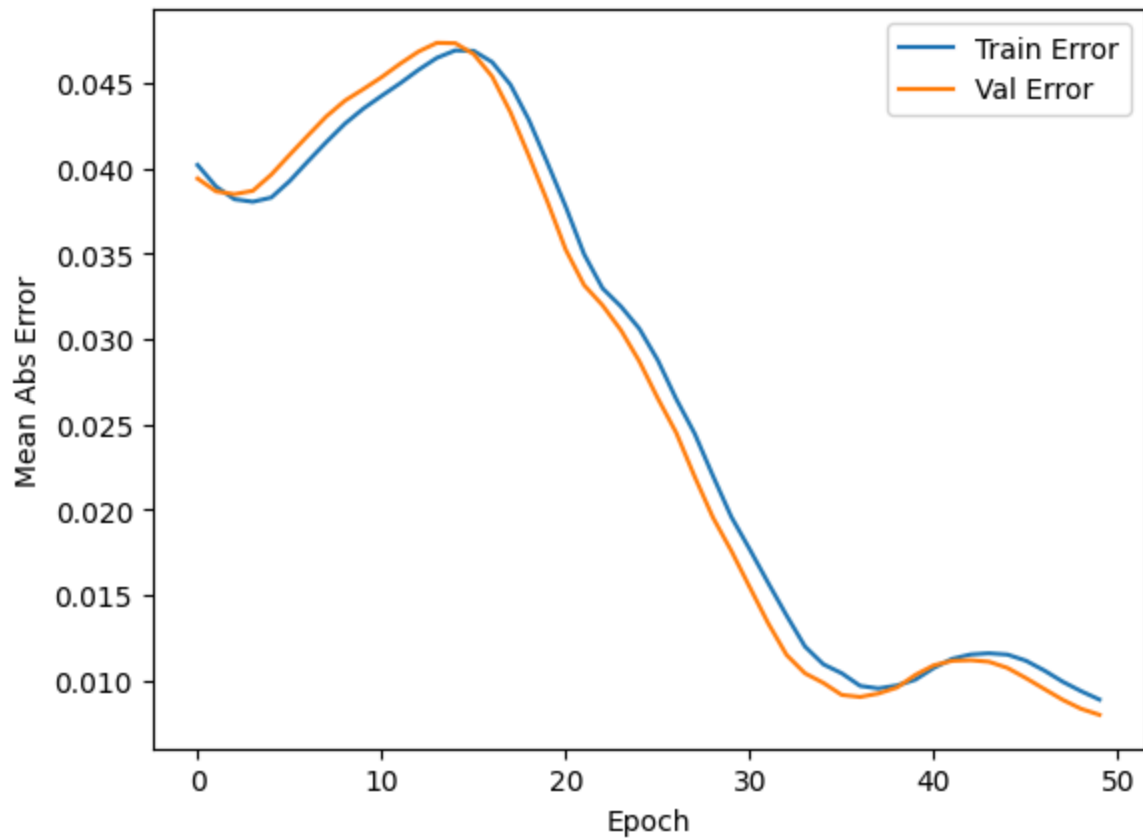
La línea `modo = 'normal'` establece la variable `modo` con el valor `'normal'`, indicando que las funciones subsecuentes deben operar en este modo. En el contexto del código, esto significa que las funciones de procesamiento de datos, definición del modelo y entrenamiento se ajustarán para manejar los datos en un formato donde cada histograma anual es tratado de manera independiente. Además, se divide el conjunto de datos en un 80% para entrenamiento y un 20% para validación, asegurando que el modelo se entrene en una muestra representativa de los datos y se valide en una porción separada para evaluar su rendimiento y generalización.

```
In [ ]: # Elegir el modo: 'normal' o 'matricial'
        modo = 'normal'
        # Dividir los datos y entrenar el modelo
        X_train, X_val, Y_train, Y_val = dividir_datos(histograms, modo=modo, porcentaje_en
        # Entrena y valida el modelo, con validacion cruzada
        model = entrenar_y_validar(X_train, Y_train, modo=modo)
        # Predecir y graficar
        predecir_y_graficar(model, X_val, histograms, modo=modo)
        # Evaluación final
        val_loss, val_mae = model.evaluate(X_val, Y_val)
        print(f"Final MAE (Validation): {val_mae}")
```


Tamaño de la matriz X_train: (28, 35, 1, 1)
Tamaño de la matriz X_val: (6, 35, 1, 1)
Tamaño de la matriz Y_train: (28, 35)
Tamaño de la matriz Y_val: (6, 35)
Entrenando fold 1...
Epoch 1/50
1/1  2s 2s/step - loss: 0.0167 - mae: 0.0402 - val_loss: 0.0159
- val_mae: 0.0394
Epoch 2/50
1/1  0s 415ms/step - loss: 0.0159 - mae: 0.0389 - val_loss: 0.0151 - val_mae: 0.0386
Epoch 3/50
1/1  0s 149ms/step - loss: 0.0151 - mae: 0.0382 - val_loss: 0.0144 - val_mae: 0.0385
Epoch 4/50
1/1  0s 140ms/step - loss: 0.0144 - mae: 0.0380 - val_loss: 0.0138 - val_mae: 0.0387
Epoch 5/50
1/1  0s 125ms/step - loss: 0.0137 - mae: 0.0383 - val_loss: 0.0132 - val_mae: 0.0396
Epoch 6/50
1/1  0s 135ms/step - loss: 0.0131 - mae: 0.0392 - val_loss: 0.0125 - val_mae: 0.0408
Epoch 7/50
1/1  0s 146ms/step - loss: 0.0125 - mae: 0.0404 - val_loss: 0.0119 - val_mae: 0.0419
Epoch 8/50
1/1  0s 111ms/step - loss: 0.0119 - mae: 0.0415 - val_loss: 0.0113 - val_mae: 0.0430
Epoch 9/50
1/1  0s 155ms/step - loss: 0.0113 - mae: 0.0426 - val_loss: 0.0107 - val_mae: 0.0439
Epoch 10/50
1/1  0s 114ms/step - loss: 0.0107 - mae: 0.0435 - val_loss: 0.0100 - val_mae: 0.0446
Epoch 11/50
1/1  0s 123ms/step - loss: 0.0100 - mae: 0.0442 - val_loss: 0.0094 - val_mae: 0.0453
Epoch 12/50
1/1  0s 96ms/step - loss: 0.0094 - mae: 0.0450 - val_loss: 0.0087 - val_mae: 0.0461
Epoch 13/50
1/1  0s 129ms/step - loss: 0.0087 - mae: 0.0457 - val_loss: 0.0081 - val_mae: 0.0468
Epoch 14/50
1/1  0s 141ms/step - loss: 0.0081 - mae: 0.0465 - val_loss: 0.0074 - val_mae: 0.0473
Epoch 15/50
1/1  0s 94ms/step - loss: 0.0074 - mae: 0.0469 - val_loss: 0.0067 - val_mae: 0.0473
Epoch 16/50
1/1  0s 158ms/step - loss: 0.0067 - mae: 0.0469 - val_loss: 0.0061 - val_mae: 0.0467
Epoch 17/50
1/1  0s 137ms/step - loss: 0.0061 - mae: 0.0462 - val_loss: 0.0054 - val_mae: 0.0454


Epoch 18/50
1/1  0s 86ms/step - loss: 0.0054 - mae: 0.0449 - val_loss: 0.0048 - val_mae: 0.0433
Epoch 19/50
1/1  0s 145ms/step - loss: 0.0048 - mae: 0.0428 - val_loss: 0.0042 - val_mae: 0.0407
Epoch 20/50
1/1  0s 99ms/step - loss: 0.0042 - mae: 0.0403 - val_loss: 0.0036 - val_mae: 0.0380
Epoch 21/50
1/1  0s 108ms/step - loss: 0.0036 - mae: 0.0377 - val_loss: 0.0031 - val_mae: 0.0352
Epoch 22/50
1/1  0s 152ms/step - loss: 0.0032 - mae: 0.0350 - val_loss: 0.0027 - val_mae: 0.0331
Epoch 23/50
1/1  0s 114ms/step - loss: 0.0027 - mae: 0.0330 - val_loss: 0.0023 - val_mae: 0.0320
Epoch 24/50
1/1  0s 151ms/step - loss: 0.0023 - mae: 0.0319 - val_loss: 0.0020 - val_mae: 0.0305
Epoch 25/50
1/1  0s 114ms/step - loss: 0.0020 - mae: 0.0306 - val_loss: 0.0017 - val_mae: 0.0287
Epoch 26/50
1/1  0s 141ms/step - loss: 0.0017 - mae: 0.0288 - val_loss: 0.0014 - val_mae: 0.0265
Epoch 27/50
1/1  0s 280ms/step - loss: 0.0014 - mae: 0.0265 - val_loss: 0.0011 - val_mae: 0.0245
Epoch 28/50
1/1  0s 108ms/step - loss: 0.0011 - mae: 0.0245 - val_loss: 8.9483e-04 - val_mae: 0.0219
Epoch 29/50
1/1  0s 110ms/step - loss: 8.9473e-04 - mae: 0.0220 - val_loss: 6.9315e-04 - val_mae: 0.0195
Epoch 30/50
1/1  0s 63ms/step - loss: 6.9653e-04 - mae: 0.0196 - val_loss: 5.2941e-04 - val_mae: 0.0176
Epoch 31/50
1/1  0s 97ms/step - loss: 5.3674e-04 - mae: 0.0177 - val_loss: 4.0023e-04 - val_mae: 0.0155
Epoch 32/50
1/1  0s 81ms/step - loss: 4.1214e-04 - mae: 0.0157 - val_loss: 3.0206e-04 - val_mae: 0.0133
Epoch 33/50
1/1  0s 63ms/step - loss: 3.1879e-04 - mae: 0.0138 - val_loss: 2.3173e-04 - val_mae: 0.0115
Epoch 34/50
1/1  0s 66ms/step - loss: 2.5310e-04 - mae: 0.0120 - val_loss: 1.8546e-04 - val_mae: 0.0104
Epoch 35/50
1/1  0s 64ms/step - loss: 2.1087e-04 - mae: 0.0109 - val_loss: 1.5900e-04 - val_mae: 0.0098
Epoch 36/50
1/1  0s 139ms/step - loss: 1.8767e-04 - mae: 0.0104 - val_loss:

1.4868e-04 - val_mae: 0.0091
Epoch 37/50
1/1  0s 136ms/step - loss: 1.7966e-04 - mae: 0.0097 - val_loss: 1.5123e-04 - val_mae: 0.0090
Epoch 38/50
1/1  0s 71ms/step - loss: 1.8350e-04 - mae: 0.0095 - val_loss: 1.6300e-04 - val_mae: 0.0092
Epoch 39/50
1/1  0s 83ms/step - loss: 1.9561e-04 - mae: 0.0097 - val_loss: 1.8002e-04 - val_mae: 0.0096
Epoch 40/50
1/1  0s 70ms/step - loss: 2.1222e-04 - mae: 0.0100 - val_loss: 1.9847e-04 - val_mae: 0.0103
Epoch 41/50
1/1  0s 62ms/step - loss: 2.2975e-04 - mae: 0.0107 - val_loss: 2.1490e-04 - val_mae: 0.0109
Epoch 42/50
1/1  0s 98ms/step - loss: 2.4507e-04 - mae: 0.0113 - val_loss: 2.2655e-04 - val_mae: 0.0111
Epoch 43/50
1/1  0s 115ms/step - loss: 2.5563e-04 - mae: 0.0115 - val_loss: 2.3172e-04 - val_mae: 0.0112
Epoch 44/50
1/1  0s 63ms/step - loss: 2.5990e-04 - mae: 0.0116 - val_loss: 2.3011e-04 - val_mae: 0.0111
Epoch 45/50
1/1  0s 135ms/step - loss: 2.5763e-04 - mae: 0.0115 - val_loss: 2.2263e-04 - val_mae: 0.0107
Epoch 46/50
1/1  0s 68ms/step - loss: 2.4968e-04 - mae: 0.0112 - val_loss: 2.1089e-04 - val_mae: 0.0101
Epoch 47/50
1/1  0s 63ms/step - loss: 2.3756e-04 - mae: 0.0106 - val_loss: 1.9662e-04 - val_mae: 0.0095
Epoch 48/50
1/1  0s 64ms/step - loss: 2.2297e-04 - mae: 0.0099 - val_loss: 1.8123e-04 - val_mae: 0.0089
Epoch 49/50
1/1  0s 70ms/step - loss: 2.0726e-04 - mae: 0.0094 - val_loss: 1.6545e-04 - val_mae: 0.0083
Epoch 50/50
1/1  0s 82ms/step - loss: 1.9112e-04 - mae: 0.0089 - val_loss: 1.4958e-04 - val_mae: 0.0080




Entrenando fold 2...


Epoch 1/50

1/1  2s 2s/step - loss: 0.0165 - mae: 0.0370 - val_loss: 0.0152
- val_mae: 0.0345


Epoch 2/50

1/1  0s 70ms/step - loss: 0.0156 - mae: 0.0347 - val_loss: 0.014
4 - val_mae: 0.0335


Epoch 3/50

1/1  0s 87ms/step - loss: 0.0148 - mae: 0.0336 - val_loss: 0.013
7 - val_mae: 0.0333


Epoch 4/50

1/1  0s 112ms/step - loss: 0.0142 - mae: 0.0335 - val_loss: 0.01
31 - val_mae: 0.0337


Epoch 5/50

1/1  0s 62ms/step - loss: 0.0136 - mae: 0.0339 - val_loss: 0.012
5 - val_mae: 0.0344


Epoch 6/50

1/1  0s 68ms/step - loss: 0.0130 - mae: 0.0348 - val_loss: 0.011
9 - val_mae: 0.0354


Epoch 7/50

1/1  0s 133ms/step - loss: 0.0124 - mae: 0.0357 - val_loss: 0.01
13 - val_mae: 0.0363


Epoch 8/50

1/1  0s 60ms/step - loss: 0.0117 - mae: 0.0366 - val_loss: 0.010
6 - val_mae: 0.0371


Epoch 9/50

1/1  0s 140ms/step - loss: 0.0110 - mae: 0.0374 - val_loss: 0.00
99 - val_mae: 0.0379


Epoch 10/50

1/1  0s 64ms/step - loss: 0.0103 - mae: 0.0382 - val_loss: 0.009
2 - val_mae: 0.0386


Epoch 11/50

1/1  0s 59ms/step - loss: 0.0095 - mae: 0.0389 - val_loss: 0.008
4 - val_mae: 0.0392


Epoch 12/50

1/1  0s 149ms/step - loss: 0.0088 - mae: 0.0395 - val_loss: 0.00
77 - val_mae: 0.0401


Epoch 13/50

1/1  0s 132ms/step - loss: 0.0080 - mae: 0.0403 - val_loss: 0.00
70 - val_mae: 0.0410


Epoch 14/50

1/1  0s 135ms/step - loss: 0.0073 - mae: 0.0412 - val_loss: 0.00
64 - val_mae: 0.0417


Epoch 15/50

1/1  0s 133ms/step - loss: 0.0067 - mae: 0.0420 - val_loss: 0.00
58 - val_mae: 0.0420


Epoch 16/50

1/1  0s 63ms/step - loss: 0.0060 - mae: 0.0425 - val_loss: 0.005
3 - val_mae: 0.0420

Epoch 17/50











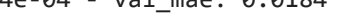








1/1  0s 62ms/step - loss: 0.0055 - mae: 0.0426 - val_loss: 0.004
8 - val_mae: 0.0419

Epoch 18/50














1/1  0s 64ms/step - loss: 0.0050 - mae: 0.0424 - val_loss: 0.004
4 - val_mae: 0.0410

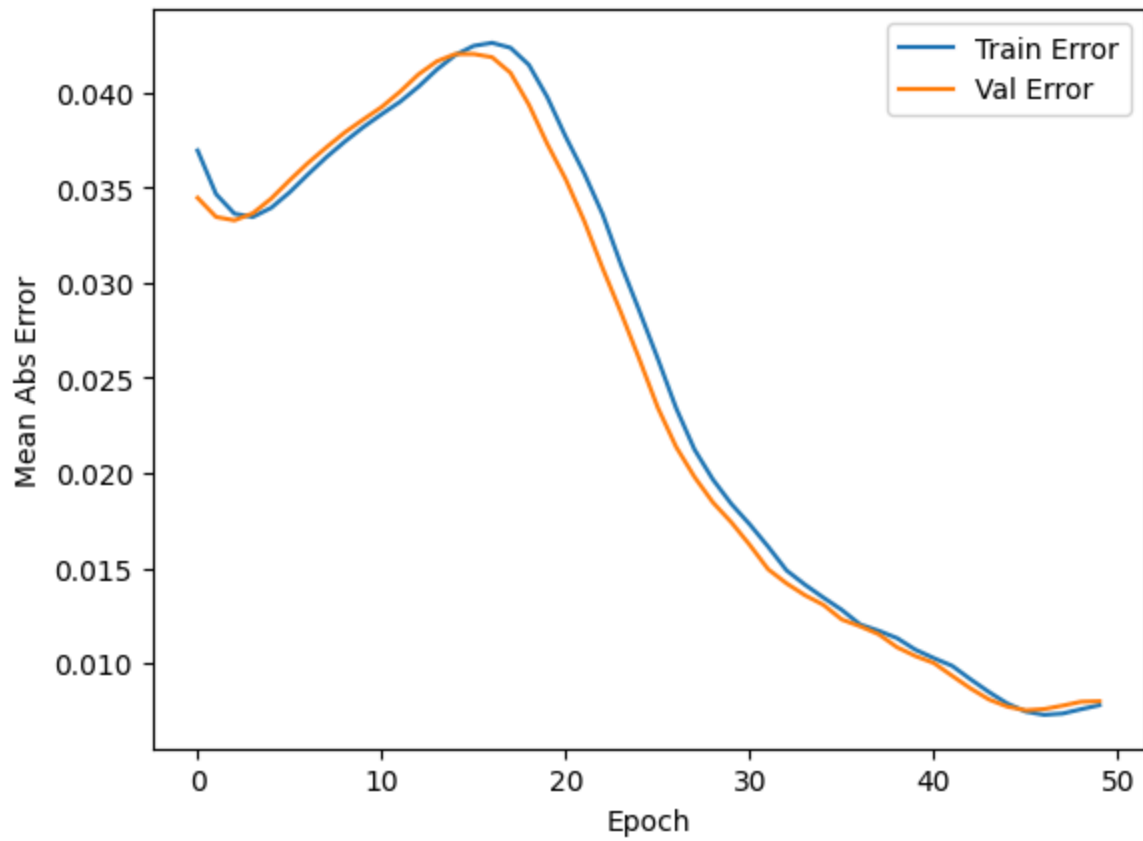
Epoch 19/50

```

1/1  0s 62ms/step - loss: 0.0046 - mae: 0.0415 - val_loss: 0.004
0 - val_mae: 0.0394
Epoch 20/50
1/1  0s 67ms/step - loss: 0.0041 - mae: 0.0398 - val_loss: 0.003
6 - val_mae: 0.0373
Epoch 21/50
1/1  0s 61ms/step - loss: 0.0037 - mae: 0.0377 - val_loss: 0.003
2 - val_mae: 0.0354
Epoch 22/50
1/1  0s 64ms/step - loss: 0.0033 - mae: 0.0358 - val_loss: 0.002
8 - val_mae: 0.0333
Epoch 23/50
1/1  0s 61ms/step - loss: 0.0029 - mae: 0.0336 - val_loss: 0.002
4 - val_mae: 0.0308
Epoch 24/50
1/1  0s 79ms/step - loss: 0.0025 - mae: 0.0310 - val_loss: 0.002
1 - val_mae: 0.0284
Epoch 25/50
1/1  0s 137ms/step - loss: 0.0021 - mae: 0.0285 - val_loss: 0.00
17 - val_mae: 0.0260
Epoch 26/50
1/1  0s 131ms/step - loss: 0.0018 - mae: 0.0260 - val_loss: 0.00
14 - val_mae: 0.0234
Epoch 27/50
1/1  0s 64ms/step - loss: 0.0015 - mae: 0.0234 - val_loss: 0.001
2 - val_mae: 0.0214
Epoch 28/50
1/1  0s 61ms/step - loss: 0.0012 - mae: 0.0212 - val_loss: 0.001
0 - val_mae: 0.0198
Epoch 29/50
1/1  0s 80ms/step - loss: 0.0010 - mae: 0.0197 - val_loss: 8.518
4e-04 - val_mae: 0.0184
Epoch 30/50
1/1  0s 121ms/step - loss: 8.6047e-04 - mae: 0.0184 - val_loss:
7.1944e-04 - val_mae: 0.0174
Epoch 31/50
1/1  0s 62ms/step - loss: 7.2498e-04 - mae: 0.0173 - val_loss:
6.0960e-04 - val_mae: 0.0162
Epoch 32/50
1/1  0s 62ms/step - loss: 6.1137e-04 - mae: 0.0161 - val_loss:
5.1837e-04 - val_mae: 0.0149
Epoch 33/50
1/1  0s 65ms/step - loss: 5.1562e-04 - mae: 0.0149 - val_loss:
4.4317e-04 - val_mae: 0.0142
Epoch 34/50
1/1  0s 65ms/step - loss: 4.3512e-04 - mae: 0.0141 - val_loss:
3.8222e-04 - val_mae: 0.0136
Epoch 35/50
1/1  0s 90ms/step - loss: 3.6826e-04 - mae: 0.0134 - val_loss:
3.3507e-04 - val_mae: 0.0131
Epoch 36/50
1/1  0s 87ms/step - loss: 3.1469e-04 - mae: 0.0128 - val_loss:
2.9833e-04 - val_mae: 0.0123
Epoch 37/50
1/1  0s 137ms/step - loss: 2.7165e-04 - mae: 0.0120 - val_loss:
2.7692e-04 - val_mae: 0.0119


```

Epoch 38/50
1/1  0s 69ms/step - loss: 2.4395e-04 - mae: 0.0117 - val_loss: 2.6781e-04 - val_mae: 0.0115
Epoch 39/50
1/1  0s 63ms/step - loss: 2.2902e-04 - mae: 0.0113 - val_loss: 2.6236e-04 - val_mae: 0.0108
Epoch 40/50
1/1  0s 69ms/step - loss: 2.1887e-04 - mae: 0.0107 - val_loss: 2.5265e-04 - val_mae: 0.0104
Epoch 41/50
1/1  0s 133ms/step - loss: 2.0616e-04 - mae: 0.0103 - val_loss: 2.3663e-04 - val_mae: 0.0100
Epoch 42/50
1/1  0s 82ms/step - loss: 1.8907e-04 - mae: 0.0099 - val_loss: 2.1643e-04 - val_mae: 0.0093
Epoch 43/50
1/1  0s 61ms/step - loss: 1.6953e-04 - mae: 0.0091 - val_loss: 1.9591e-04 - val_mae: 0.0087
Epoch 44/50
1/1  0s 59ms/step - loss: 1.5100e-04 - mae: 0.0085 - val_loss: 1.7896e-04 - val_mae: 0.0081
Epoch 45/50
1/1  0s 139ms/step - loss: 1.3690e-04 - mae: 0.0079 - val_loss: 1.6821e-04 - val_mae: 0.0077
Epoch 46/50
1/1  0s 91ms/step - loss: 1.2930e-04 - mae: 0.0074 - val_loss: 1.6396e-04 - val_mae: 0.0075
Epoch 47/50
1/1  0s 132ms/step - loss: 1.2786e-04 - mae: 0.0073 - val_loss: 1.6424e-04 - val_mae: 0.0076
Epoch 48/50
1/1  0s 68ms/step - loss: 1.3036e-04 - mae: 0.0073 - val_loss: 1.6654e-04 - val_mae: 0.0078
Epoch 49/50
1/1  0s 63ms/step - loss: 1.3413e-04 - mae: 0.0076 - val_loss: 1.6859e-04 - val_mae: 0.0080
Epoch 50/50
1/1  0s 63ms/step - loss: 1.3691e-04 - mae: 0.0078 - val_loss: 1.6883e-04 - val_mae: 0.0080




Entrenando fold 3...


Epoch 1/50

1/1  2s 2s/step - loss: 0.0178 - mae: 0.0373 - val_loss: 0.0172
- val_mae: 0.0350


Epoch 2/50

1/1  0s 365ms/step - loss: 0.0168 - mae: 0.0348 - val_loss: 0.0163 - val_mae: 0.0334


Epoch 3/50

1/1  0s 110ms/step - loss: 0.0159 - mae: 0.0331 - val_loss: 0.0156 - val_mae: 0.0327


Epoch 4/50

1/1  0s 164ms/step - loss: 0.0152 - mae: 0.0325 - val_loss: 0.0150 - val_mae: 0.0326


Epoch 5/50

1/1  0s 108ms/step - loss: 0.0146 - mae: 0.0323 - val_loss: 0.0145 - val_mae: 0.0326


Epoch 6/50

1/1  0s 95ms/step - loss: 0.0142 - mae: 0.0323 - val_loss: 0.0140 - val_mae: 0.0327


Epoch 7/50

1/1  0s 86ms/step - loss: 0.0137 - mae: 0.0324 - val_loss: 0.0135 - val_mae: 0.0328


Epoch 8/50

1/1  0s 136ms/step - loss: 0.0132 - mae: 0.0326 - val_loss: 0.0131 - val_mae: 0.0332


Epoch 9/50

1/1  0s 176ms/step - loss: 0.0128 - mae: 0.0330 - val_loss: 0.0126 - val_mae: 0.0338


Epoch 10/50

1/1  0s 262ms/step - loss: 0.0123 - mae: 0.0334 - val_loss: 0.0121 - val_mae: 0.0345


Epoch 11/50

1/1  0s 96ms/step - loss: 0.0119 - mae: 0.0340 - val_loss: 0.0116 - val_mae: 0.0353


Epoch 12/50

1/1  0s 146ms/step - loss: 0.0114 - mae: 0.0348 - val_loss: 0.0111 - val_mae: 0.0363


Epoch 13/50

1/1  0s 138ms/step - loss: 0.0108 - mae: 0.0356 - val_loss: 0.0105 - val_mae: 0.0372


Epoch 14/50

1/1  0s 107ms/step - loss: 0.0103 - mae: 0.0365 - val_loss: 0.0100 - val_mae: 0.0380


Epoch 15/50

1/1  0s 158ms/step - loss: 0.0097 - mae: 0.0373 - val_loss: 0.0094 - val_mae: 0.0390


Epoch 16/50

1/1  0s 116ms/step - loss: 0.0092 - mae: 0.0384 - val_loss: 0.0088 - val_mae: 0.0402

Epoch 17/50















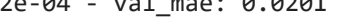




1/1  0s 94ms/step - loss: 0.0086 - mae: 0.0395 - val_loss: 0.0083 - val_mae: 0.0412

Epoch 18/50














1/1  0s 90ms/step - loss: 0.0081 - mae: 0.0406 - val_loss: 0.0077 - val_mae: 0.0423

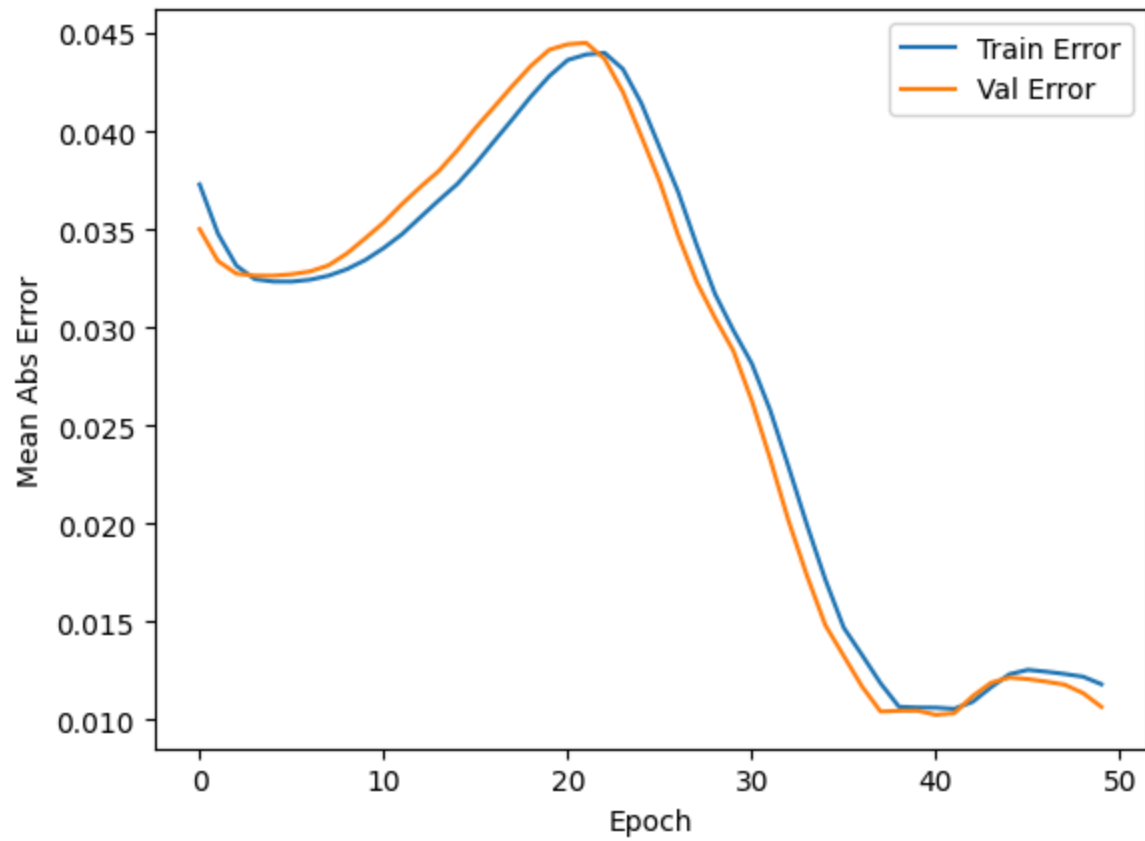
Epoch 19/50

```

1/1  0s 176ms/step - loss: 0.0075 - mae: 0.0418 - val_loss: 0.00
72 - val_mae: 0.0433
Epoch 20/50
1/1  0s 98ms/step - loss: 0.0070 - mae: 0.0428 - val_loss: 0.006
7 - val_mae: 0.0442
Epoch 21/50
1/1  0s 97ms/step - loss: 0.0065 - mae: 0.0436 - val_loss: 0.006
2 - val_mae: 0.0444
Epoch 22/50
1/1  0s 118ms/step - loss: 0.0060 - mae: 0.0439 - val_loss: 0.00
57 - val_mae: 0.0445
Epoch 23/50
1/1  0s 128ms/step - loss: 0.0056 - mae: 0.0440 - val_loss: 0.00
52 - val_mae: 0.0437
Epoch 24/50
1/1  0s 76ms/step - loss: 0.0050 - mae: 0.0432 - val_loss: 0.004
7 - val_mae: 0.0420
Epoch 25/50
1/1  0s 83ms/step - loss: 0.0045 - mae: 0.0414 - val_loss: 0.004
1 - val_mae: 0.0397
Epoch 26/50
1/1  0s 71ms/step - loss: 0.0040 - mae: 0.0392 - val_loss: 0.003
6 - val_mae: 0.0374
Epoch 27/50
1/1  0s 64ms/step - loss: 0.0035 - mae: 0.0369 - val_loss: 0.003
0 - val_mae: 0.0347
Epoch 28/50
1/1  0s 136ms/step - loss: 0.0029 - mae: 0.0342 - val_loss: 0.00
26 - val_mae: 0.0323
Epoch 29/50
1/1  0s 64ms/step - loss: 0.0025 - mae: 0.0317 - val_loss: 0.002
2 - val_mae: 0.0305
Epoch 30/50
1/1  0s 66ms/step - loss: 0.0021 - mae: 0.0298 - val_loss: 0.001
8 - val_mae: 0.0288
Epoch 31/50
1/1  0s 60ms/step - loss: 0.0017 - mae: 0.0282 - val_loss: 0.001
5 - val_mae: 0.0263
Epoch 32/50
1/1  0s 68ms/step - loss: 0.0014 - mae: 0.0258 - val_loss: 0.001
2 - val_mae: 0.0233
Epoch 33/50
1/1  0s 62ms/step - loss: 0.0011 - mae: 0.0229 - val_loss: 8.859
2e-04 - val_mae: 0.0201
Epoch 34/50
1/1  0s 70ms/step - loss: 8.6429e-04 - mae: 0.0199 - val_loss:
6.6171e-04 - val_mae: 0.0173
Epoch 35/50
1/1  0s 144ms/step - loss: 6.5085e-04 - mae: 0.0171 - val_loss:
4.8653e-04 - val_mae: 0.0148
Epoch 36/50
1/1  0s 72ms/step - loss: 4.8566e-04 - mae: 0.0147 - val_loss:
3.5928e-04 - val_mae: 0.0132
Epoch 37/50
1/1  0s 78ms/step - loss: 3.6760e-04 - mae: 0.0133 - val_loss:
2.7286e-04 - val_mae: 0.0117


```

Epoch 38/50
1/1  0s 129ms/step - loss: 2.8951e-04 - mae: 0.0118 - val_loss: 2.2020e-04 - val_mae: 0.0104
Epoch 39/50
1/1  0s 61ms/step - loss: 2.4407e-04 - mae: 0.0106 - val_loss: 1.9458e-04 - val_mae: 0.0104
Epoch 40/50
1/1  0s 62ms/step - loss: 2.2434e-04 - mae: 0.0106 - val_loss: 1.8964e-04 - val_mae: 0.0104
Epoch 41/50
1/1  0s 62ms/step - loss: 2.2392e-04 - mae: 0.0106 - val_loss: 2.0091e-04 - val_mae: 0.0102
Epoch 42/50
1/1  0s 68ms/step - loss: 2.3839e-04 - mae: 0.0105 - val_loss: 2.2276e-04 - val_mae: 0.0103
Epoch 43/50
1/1  0s 62ms/step - loss: 2.6237e-04 - mae: 0.0109 - val_loss: 2.4733e-04 - val_mae: 0.0112
Epoch 44/50
1/1  0s 57ms/step - loss: 2.8823e-04 - mae: 0.0116 - val_loss: 2.6791e-04 - val_mae: 0.0119
Epoch 45/50
1/1  0s 147ms/step - loss: 3.0936e-04 - mae: 0.0123 - val_loss: 2.8029e-04 - val_mae: 0.0121
Epoch 46/50
1/1  0s 75ms/step - loss: 3.2155e-04 - mae: 0.0125 - val_loss: 2.8209e-04 - val_mae: 0.0121
Epoch 47/50
1/1  0s 93ms/step - loss: 3.2253e-04 - mae: 0.0124 - val_loss: 2.7385e-04 - val_mae: 0.0119
Epoch 48/50
1/1  0s 124ms/step - loss: 3.1288e-04 - mae: 0.0123 - val_loss: 2.5799e-04 - val_mae: 0.0118
Epoch 49/50
1/1  0s 59ms/step - loss: 2.9525e-04 - mae: 0.0122 - val_loss: 2.3711e-04 - val_mae: 0.0113
Epoch 50/50
1/1  0s 64ms/step - loss: 2.7245e-04 - mae: 0.0118 - val_loss: 2.1358e-04 - val_mae: 0.0106




Entrenando fold 4...


Epoch 1/50

1/1  2s 2s/step - loss: 0.0166 - mae: 0.0389 - val_loss: 0.0149
- val_mae: 0.0361


Epoch 2/50

1/1  0s 78ms/step - loss: 0.0158 - mae: 0.0372 - val_loss: 0.014
3 - val_mae: 0.0349


Epoch 3/50

1/1  0s 62ms/step - loss: 0.0152 - mae: 0.0360 - val_loss: 0.013
7 - val_mae: 0.0346


Epoch 4/50

1/1  0s 68ms/step - loss: 0.0146 - mae: 0.0357 - val_loss: 0.013
2 - val_mae: 0.0353


Epoch 5/50

1/1  0s 60ms/step - loss: 0.0141 - mae: 0.0361 - val_loss: 0.012
7 - val_mae: 0.0362


Epoch 6/50

1/1  0s 69ms/step - loss: 0.0135 - mae: 0.0371 - val_loss: 0.012
2 - val_mae: 0.0371


Epoch 7/50

1/1  0s 60ms/step - loss: 0.0130 - mae: 0.0380 - val_loss: 0.011
6 - val_mae: 0.0376


Epoch 8/50

1/1  0s 73ms/step - loss: 0.0124 - mae: 0.0386 - val_loss: 0.011
1 - val_mae: 0.0381


Epoch 9/50

1/1  0s 80ms/step - loss: 0.0118 - mae: 0.0389 - val_loss: 0.010
5 - val_mae: 0.0385


Epoch 10/50

1/1  0s 72ms/step - loss: 0.0112 - mae: 0.0393 - val_loss: 0.009
8 - val_mae: 0.0387


Epoch 11/50

1/1  0s 63ms/step - loss: 0.0105 - mae: 0.0396 - val_loss: 0.009
2 - val_mae: 0.0389


Epoch 12/50

1/1  0s 69ms/step - loss: 0.0098 - mae: 0.0397 - val_loss: 0.008
5 - val_mae: 0.0391


Epoch 13/50

1/1  0s 78ms/step - loss: 0.0091 - mae: 0.0399 - val_loss: 0.007
8 - val_mae: 0.0396


Epoch 14/50

1/1  0s 120ms/step - loss: 0.0084 - mae: 0.0403 - val_loss: 0.00
71 - val_mae: 0.0400


Epoch 15/50

1/1  0s 67ms/step - loss: 0.0076 - mae: 0.0407 - val_loss: 0.006
4 - val_mae: 0.0404


Epoch 16/50

1/1  0s 62ms/step - loss: 0.0068 - mae: 0.0412 - val_loss: 0.005
6 - val_mae: 0.0406


Epoch 17/50


1/1  0s 64ms/step - loss: 0.0060 - mae: 0.0414 - val_loss: 0.004
9 - val_mae: 0.0403


Epoch 18/50


1/1  0s 63ms/step - loss: 0.0053 - mae: 0.0410 - val_loss: 0.004
2 - val_mae: 0.0390


Epoch 19/50


1/1  0s 66ms/step - loss: 0.0045 - mae: 0.0397 - val_loss: 0.0035 - val_mae: 0.0370
Epoch 20/50


1/1  0s 66ms/step - loss: 0.0038 - mae: 0.0377 - val_loss: 0.0029 - val_mae: 0.0345
Epoch 21/50


1/1  0s 77ms/step - loss: 0.0031 - mae: 0.0352 - val_loss: 0.0024 - val_mae: 0.0320
Epoch 22/50


1/1  0s 136ms/step - loss: 0.0025 - mae: 0.0325 - val_loss: 0.0019 - val_mae: 0.0294
Epoch 23/50


1/1  0s 79ms/step - loss: 0.0020 - mae: 0.0296 - val_loss: 0.0014 - val_mae: 0.0268
Epoch 24/50


1/1  0s 64ms/step - loss: 0.0015 - mae: 0.0267 - val_loss: 0.0011 - val_mae: 0.0235
Epoch 25/50


1/1  0s 85ms/step - loss: 0.0011 - mae: 0.0233 - val_loss: 7.9250e-04 - val_mae: 0.0208
Epoch 26/50


1/1  0s 66ms/step - loss: 8.3423e-04 - mae: 0.0207 - val_loss: 6.2134e-04 - val_mae: 0.0193
Epoch 27/50


1/1  0s 64ms/step - loss: 6.5344e-04 - mae: 0.0195 - val_loss: 5.4054e-04 - val_mae: 0.0181
Epoch 28/50


1/1  0s 68ms/step - loss: 5.6515e-04 - mae: 0.0185 - val_loss: 5.1398e-04 - val_mae: 0.0175
Epoch 29/50


1/1  0s 66ms/step - loss: 5.3002e-04 - mae: 0.0176 - val_loss: 5.0966e-04 - val_mae: 0.0163
Epoch 30/50


1/1  0s 61ms/step - loss: 5.1526e-04 - mae: 0.0162 - val_loss: 5.1091e-04 - val_mae: 0.0161
Epoch 31/50


1/1  0s 65ms/step - loss: 5.0569e-04 - mae: 0.0161 - val_loss: 5.0766e-04 - val_mae: 0.0165
Epoch 32/50


1/1  0s 62ms/step - loss: 4.9318e-04 - mae: 0.0164 - val_loss: 4.9292e-04 - val_mae: 0.0169
Epoch 33/50


1/1  0s 142ms/step - loss: 4.7174e-04 - mae: 0.0166 - val_loss: 4.6518e-04 - val_mae: 0.0167
Epoch 34/50


1/1  0s 82ms/step - loss: 4.4022e-04 - mae: 0.0161 - val_loss: 4.2864e-04 - val_mae: 0.0159
Epoch 35/50


1/1  0s 67ms/step - loss: 4.0258e-04 - mae: 0.0153 - val_loss: 3.8893e-04 - val_mae: 0.0149
Epoch 36/50


1/1  0s 61ms/step - loss: 3.6571e-04 - mae: 0.0146 - val_loss: 3.5418e-04 - val_mae: 0.0138
Epoch 37/50


1/1  0s 149ms/step - loss: 3.3521e-04 - mae: 0.0137 - val_loss: 3.2830e-04 - val_mae: 0.0128


Epoch 38/50
1/1  0s 126ms/step - loss: 3.1218e-04 - mae: 0.0126 - val_loss: 3.1139e-04 - val_mae: 0.0121


Epoch 39/50
1/1  0s 69ms/step - loss: 2.9605e-04 - mae: 0.0117 - val_loss: 3.0243e-04 - val_mae: 0.0119


Epoch 40/50
1/1  0s 130ms/step - loss: 2.8589e-04 - mae: 0.0113 - val_loss: 2.9928e-04 - val_mae: 0.0120


Epoch 41/50
1/1  0s 67ms/step - loss: 2.8011e-04 - mae: 0.0115 - val_loss: 2.9831e-04 - val_mae: 0.0123


Epoch 42/50
1/1  0s 59ms/step - loss: 2.7568e-04 - mae: 0.0118 - val_loss: 2.9534e-04 - val_mae: 0.0125


Epoch 43/50
1/1  0s 76ms/step - loss: 2.6898e-04 - mae: 0.0119 - val_loss: 2.8742e-04 - val_mae: 0.0125


Epoch 44/50
1/1  0s 77ms/step - loss: 2.5744e-04 - mae: 0.0118 - val_loss: 2.7345e-04 - val_mae: 0.0123


Epoch 45/50
1/1  0s 129ms/step - loss: 2.4022e-04 - mae: 0.0115 - val_loss: 2.5408e-04 - val_mae: 0.0118

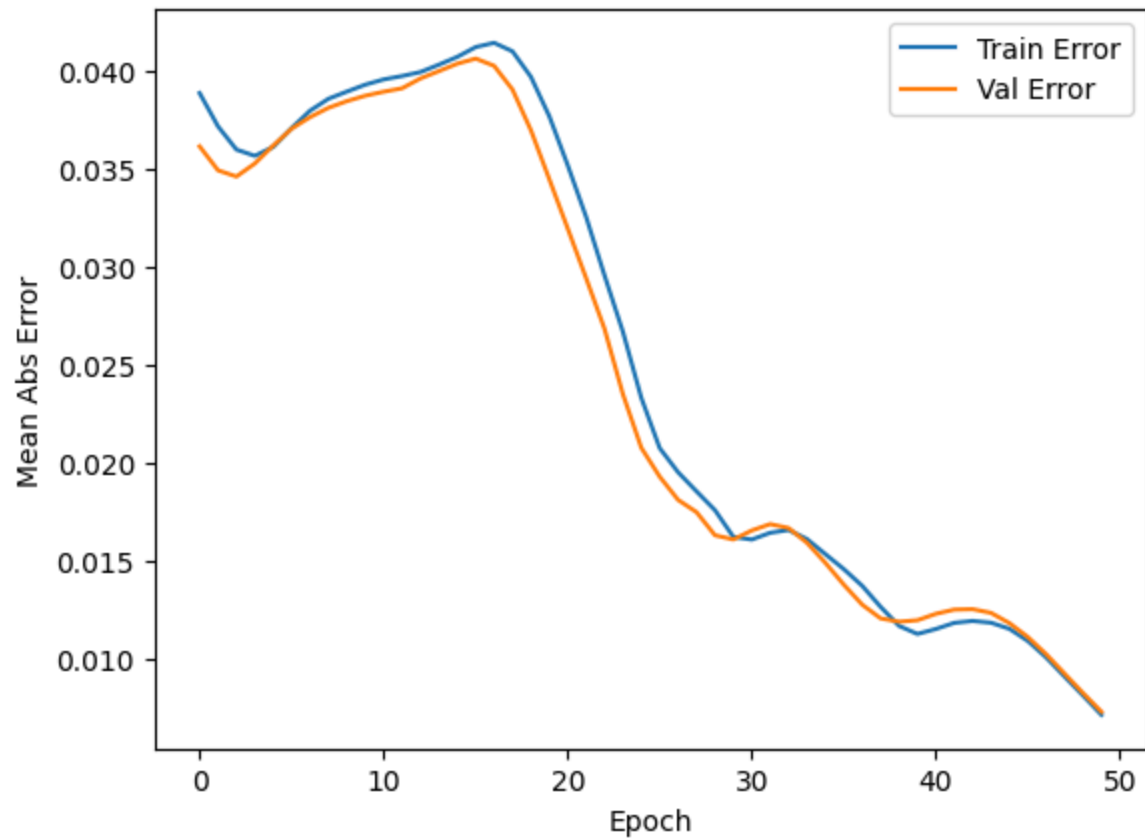
Epoch 46/50
1/1  0s 64ms/step - loss: 2.1815e-04 - mae: 0.0109 - val_loss: 2.3122e-04 - val_mae: 0.0111

Epoch 47/50
1/1  0s 143ms/step - loss: 1.9321e-04 - mae: 0.0101 - val_loss: 2.0703e-04 - val_mae: 0.0102

Epoch 48/50
1/1  0s 127ms/step - loss: 1.6774e-04 - mae: 0.0091 - val_loss: 1.8338e-04 - val_mae: 0.0092


Epoch 49/50
1/1  0s 67ms/step - loss: 1.4372e-04 - mae: 0.0081 - val_loss: 1.6145e-04 - val_mae: 0.0082

Epoch 50/50
1/1  0s 150ms/step - loss: 1.2246e-04 - mae: 0.0071 - val_loss: 1.4188e-04 - val_mae: 0.0073




Entrenando fold 5...


Epoch 1/50

1/1  2s 2s/step - loss: 0.0170 - mae: 0.0388 - val_loss: 0.0170
- val_mae: 0.0385


Epoch 2/50

1/1  0s 77ms/step - loss: 0.0159 - mae: 0.0383 - val_loss: 0.0159
- val_mae: 0.0381


Epoch 3/50

1/1  0s 167ms/step - loss: 0.0148 - mae: 0.0378 - val_loss: 0.0149
- val_mae: 0.0380


Epoch 4/50

1/1  0s 137ms/step - loss: 0.0139 - mae: 0.0378 - val_loss: 0.0141
- val_mae: 0.0386


Epoch 5/50

1/1  0s 81ms/step - loss: 0.0130 - mae: 0.0383 - val_loss: 0.0133
- val_mae: 0.0399


Epoch 6/50

1/1  0s 142ms/step - loss: 0.0123 - mae: 0.0397 - val_loss: 0.0126
- val_mae: 0.0414


Epoch 7/50

1/1  0s 171ms/step - loss: 0.0116 - mae: 0.0412 - val_loss: 0.0120
- val_mae: 0.0425


Epoch 8/50

1/1  0s 264ms/step - loss: 0.0110 - mae: 0.0424 - val_loss: 0.0113
- val_mae: 0.0436


Epoch 9/50

1/1  0s 132ms/step - loss: 0.0104 - mae: 0.0434 - val_loss: 0.0106
- val_mae: 0.0450


Epoch 10/50

1/1  0s 136ms/step - loss: 0.0097 - mae: 0.0445 - val_loss: 0.0100
- val_mae: 0.0466


Epoch 11/50

1/1  0s 152ms/step - loss: 0.0091 - mae: 0.0458 - val_loss: 0.0093
- val_mae: 0.0482


Epoch 12/50

1/1  0s 93ms/step - loss: 0.0084 - mae: 0.0472 - val_loss: 0.0086
- val_mae: 0.0494


Epoch 13/50

1/1  0s 114ms/step - loss: 0.0078 - mae: 0.0483 - val_loss: 0.0079
- val_mae: 0.0499


Epoch 14/50

1/1  0s 115ms/step - loss: 0.0072 - mae: 0.0488 - val_loss: 0.0073
- val_mae: 0.0495


Epoch 15/50

1/1  0s 103ms/step - loss: 0.0066 - mae: 0.0483 - val_loss: 0.0066
- val_mae: 0.0483


Epoch 16/50

1/1  0s 150ms/step - loss: 0.0060 - mae: 0.0471 - val_loss: 0.0059
- val_mae: 0.0467


Epoch 17/50


1/1  0s 91ms/step - loss: 0.0054 - mae: 0.0454 - val_loss: 0.0053
- val_mae: 0.0446


Epoch 18/50


1/1  0s 147ms/step - loss: 0.0047 - mae: 0.0433 - val_loss: 0.0046
- val_mae: 0.0423


Epoch 19/50


1/1  0s 138ms/step - loss: 0.0041 - mae: 0.0410 - val_loss: 0.0040 - val_mae: 0.0393
Epoch 20/50


1/1  0s 282ms/step - loss: 0.0035 - mae: 0.0381 - val_loss: 0.0034 - val_mae: 0.0360
Epoch 21/50


1/1  0s 137ms/step - loss: 0.0030 - mae: 0.0348 - val_loss: 0.0028 - val_mae: 0.0330
Epoch 22/50


1/1  0s 103ms/step - loss: 0.0025 - mae: 0.0318 - val_loss: 0.0023 - val_mae: 0.0299
Epoch 23/50


1/1  0s 126ms/step - loss: 0.0020 - mae: 0.0288 - val_loss: 0.0019 - val_mae: 0.0273
Epoch 24/50


1/1  0s 61ms/step - loss: 0.0016 - mae: 0.0260 - val_loss: 0.0016 - val_mae: 0.0260
Epoch 25/50


1/1  0s 70ms/step - loss: 0.0013 - mae: 0.0246 - val_loss: 0.0013 - val_mae: 0.0248
Epoch 26/50


1/1  0s 66ms/step - loss: 0.0011 - mae: 0.0234 - val_loss: 0.0011 - val_mae: 0.0238
Epoch 27/50


1/1  0s 132ms/step - loss: 9.5593e-04 - mae: 0.0226 - val_loss: 9.5725e-04 - val_mae: 0.0229
Epoch 28/50


1/1  0s 61ms/step - loss: 8.5093e-04 - mae: 0.0219 - val_loss: 8.5140e-04 - val_mae: 0.0220
Epoch 29/50


1/1  0s 82ms/step - loss: 7.8239e-04 - mae: 0.0210 - val_loss: 7.6524e-04 - val_mae: 0.0209
Epoch 30/50


1/1  0s 92ms/step - loss: 7.2906e-04 - mae: 0.0201 - val_loss: 6.8047e-04 - val_mae: 0.0196
Epoch 31/50


1/1  0s 124ms/step - loss: 6.7173e-04 - mae: 0.0191 - val_loss: 5.9429e-04 - val_mae: 0.0182
Epoch 32/50


1/1  0s 64ms/step - loss: 6.0398e-04 - mae: 0.0179 - val_loss: 5.0704e-04 - val_mae: 0.0166
Epoch 33/50














1/1  0s 69ms/step - loss: 5.2601e-04 - mae: 0.0166 - val_loss: 4.2559e-04 - val_mae: 0.0150
Epoch 34/50

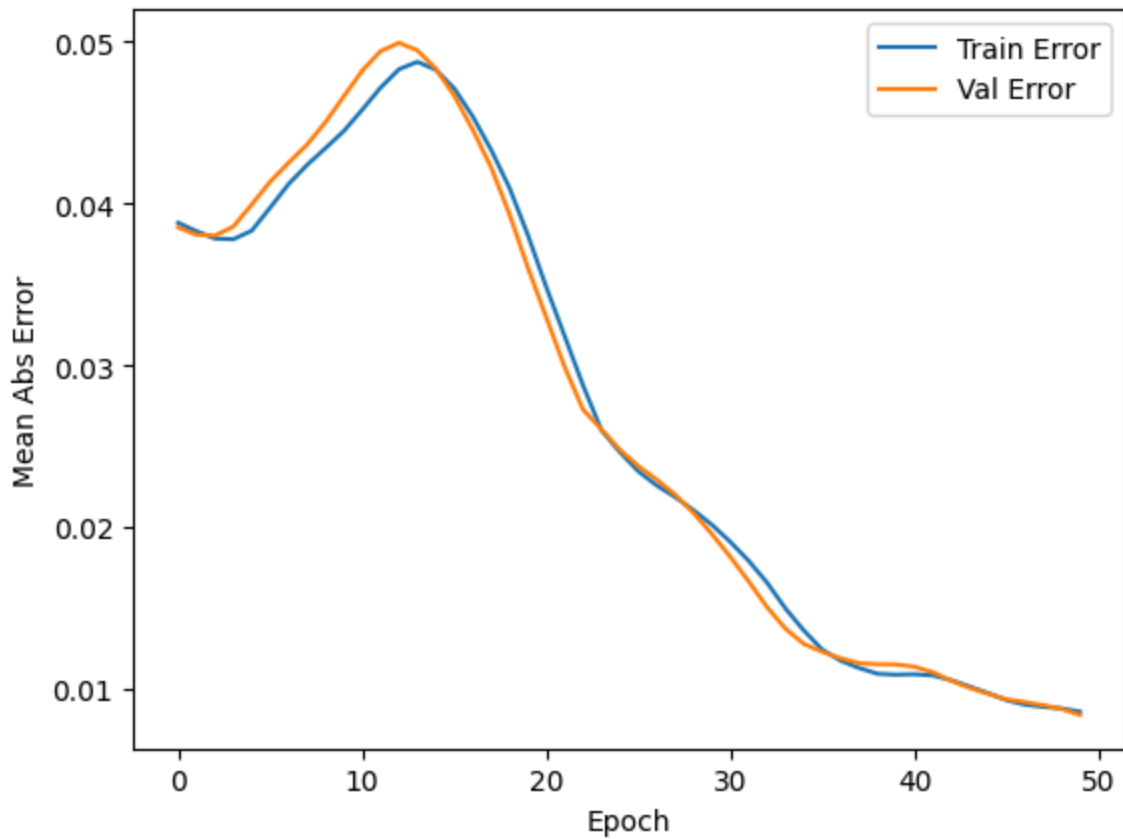
1/1  0s 153ms/step - loss: 4.4581e-04 - mae: 0.0150 - val_loss: 3.5765e-04 - val_mae: 0.0137
Epoch 35/50

1/1  0s 65ms/step - loss: 3.7302e-04 - mae: 0.0136 - val_loss: 3.0718e-04 - val_mae: 0.0128
Epoch 36/50

1/1  0s 67ms/step - loss: 3.1379e-04 - mae: 0.0124 - val_loss: 2.7348e-04 - val_mae: 0.0123
Epoch 37/50

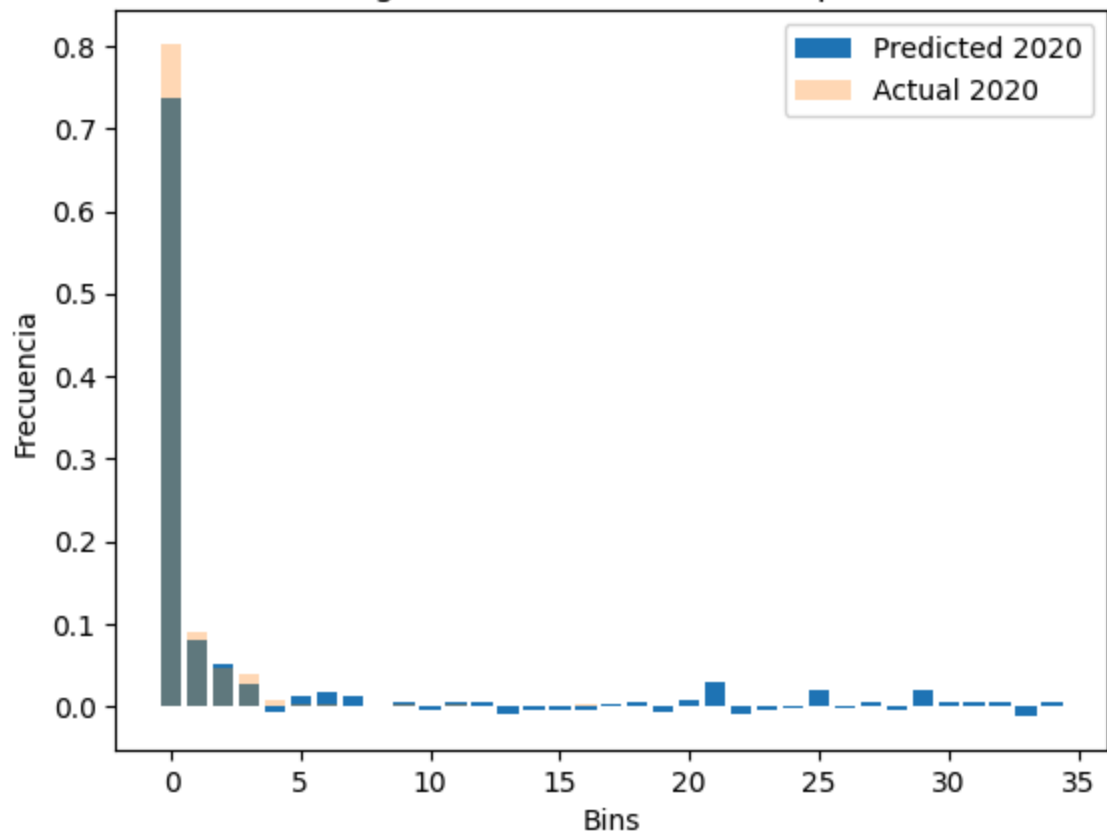
1/1  0s 66ms/step - loss: 2.6931e-04 - mae: 0.0117 - val_loss: 2.5259e-04 - val_mae: 0.0119

Epoch 38/50
1/1  0s 138ms/step - loss: 2.3764e-04 - mae: 0.0113 - val_loss: 2.4099e-04 - val_mae: 0.0116
Epoch 39/50
1/1  0s 65ms/step - loss: 2.1665e-04 - mae: 0.0109 - val_loss: 2.3606e-04 - val_mae: 0.0115
Epoch 40/50
1/1  0s 71ms/step - loss: 2.0445e-04 - mae: 0.0109 - val_loss: 2.3548e-04 - val_mae: 0.0115
Epoch 41/50
1/1  0s 101ms/step - loss: 1.9899e-04 - mae: 0.0109 - val_loss: 2.3663e-04 - val_mae: 0.0114
Epoch 42/50
1/1  0s 64ms/step - loss: 1.9776e-04 - mae: 0.0108 - val_loss: 2.3731e-04 - val_mae: 0.0110
Epoch 43/50
1/1  0s 67ms/step - loss: 1.9815e-04 - mae: 0.0105 - val_loss: 2.3594e-04 - val_mae: 0.0105
Epoch 44/50
1/1  0s 63ms/step - loss: 1.9797e-04 - mae: 0.0101 - val_loss: 2.3142e-04 - val_mae: 0.0101
Epoch 45/50
1/1  0s 66ms/step - loss: 1.9563e-04 - mae: 0.0097 - val_loss: 2.2366e-04 - val_mae: 0.0097
Epoch 46/50
1/1  0s 62ms/step - loss: 1.9048e-04 - mae: 0.0093 - val_loss: 2.1318e-04 - val_mae: 0.0093
Epoch 47/50
1/1  0s 69ms/step - loss: 1.8270e-04 - mae: 0.0090 - val_loss: 2.0084e-04 - val_mae: 0.0092
Epoch 48/50
1/1  0s 61ms/step - loss: 1.7292e-04 - mae: 0.0089 - val_loss: 1.8756e-04 - val_mae: 0.0090
Epoch 49/50
1/1  0s 140ms/step - loss: 1.6191e-04 - mae: 0.0088 - val_loss: 1.7416e-04 - val_mae: 0.0087
Epoch 50/50
1/1  0s 69ms/step - loss: 1.5039e-04 - mae: 0.0086 - val_loss: 1.6125e-04 - val_mae: 0.0084



1/1 — 0s 70ms/step

Histograma de frecuencias comparado



1/1 — 0s 24ms/step - loss: 3.2795e-04 - mae: 0.0099
Final MAE (Validation): 0.009898991324007511

Análisis de Resultados

En este análisis, se comparan dos enfoques diferentes para predecir el histograma de frecuencias para el año 2020 utilizando datos históricos de deforestación. Los enfoques comparados son el enfoque matricial y el enfoque normal.

1. Resultados del Enfoque Matricial

- **MAE Final (Validación):** 0.0040
- **Gráfico:** Se observa que la predicción en 2020 se aproxima bastante al histograma real, lo que indica que el modelo ha capturado bien la distribución subyacente en los datos.

2. Resultados del Enfoque Normal

- **MAE Final (Validación):** 0.0099
- **Gráfico:** En este caso, la predicción para el año 2020 es algo menos precisa en comparación con el enfoque matricial. Aunque la forma general de la distribución está capturada, la precisión en términos de error medio absoluto (MAE) es menor.

Conclusión

El análisis de los resultados muestra que el enfoque matricial proporciona un mejor rendimiento en la predicción del histograma de frecuencias para el año 2020, con un MAE menor (0.0040) comparado con el enfoque normal (0.0099). Este mejor desempeño puede deberse a que el enfoque matricial utiliza bloques de años consecutivos como entradas para el modelo, lo que permite capturar mejor las tendencias temporales en los datos. Esto es especialmente importante en un contexto de series temporales, donde las relaciones temporales entre los datos son clave para una predicción precisa.

En contraste, el enfoque normal, que trata cada año de manera independiente, puede perder parte de esta información temporal, lo que resulta en una menor precisión en la predicción.

Este resultado destaca la importancia de utilizar enfoques que consideren la estructura temporal y secuencial de los datos, especialmente en problemas de predicción donde los patrones históricos tienen un impacto significativo en el futuro.

Además, el uso de redes neuronales convolucionales en este contexto es particularmente relevante. Las redes convolucionales son conocidas por su capacidad para capturar patrones espaciales y temporales en los datos. En este caso, la estructura matricial permite que las convoluciones capten mejor las interacciones temporales entre los histogramas de diferentes años, lo que se traduce en una mejora en la precisión de las predicciones. Esto sugiere que las redes convolucionales, combinadas con un enfoque matricial, son

herramientas poderosas para modelar fenómenos complejos como la deforestación, donde la información histórica y secuencial es crucial para generar predicciones precisas y confiables.