

✓ Objetivo : Practica Deeplearning

El objetivo de la práctica final del módulo de Deep Learning consiste en solucionar un problema del mundo real usando las técnicas vistas en clase. En concreto, lo que se pretende es tratar de predecir el precio de habitaciones de Airbnb utilizando para ello todas las características disponibles en el dataset. El propósito final no es tener un sistema con una precisión altísima, sino que combinéis distintos tipos de características (numéricas, texto, imágenes...) y que expliquéis cómo lo habéis hecho.\

Implementación de un algoritmo predictivo para el precio de habitaciones

Módulo 1: Estimación individual de precios

A. Modelo basado en datos 1D:

Red neuronal densa: Diseño de una red neuronal con capas Dense para procesar datos tabulares. Selección del número de capas y neuronas por capa. Entrenamiento de la red neuronal con un optimizador como Adam y una función de pérdida adecuada. Características: Se utilizarán variables relevantes como tamaño de la habitación, tipo de cama, ubicación, etc. Preprocesamiento de datos categóricos y numéricos.

B. Modelo basado en imágenes:

Arquitectura: Se probarán diferentes arquitecturas CNN como:

- ResNet: Modelo robusto con buenas capacidades de aprendizaje profundo.
- VGGNet: Arquitectura simple y eficiente para clasificación de imágenes.
- EfficientNet: Modelo eficiente con buen balance entre precisión y complejidad.

Experimentos: Entrenamiento de las arquitecturas con diferentes conjuntos de datos de imágenes de habitaciones. Optimización de hiperparámetros como la tasa de aprendizaje, el número de epochs y el tamaño del batch. Evaluación del rendimiento de los modelos en conjuntos de validación y prueba. Procesamiento de imágenes: Redimensionamiento y normalización de las imágenes. Aumentación de datos para ampliar el conjunto de entrenamiento (opcional).

Módulo 2: Modelo híbrido

A. Combinación de modelos:

Fusión temprana: Concatenación de las características de las imágenes y datos 1D antes de la entrada a la red neuronal. Fusión tardía: Entrenamiento de dos redes independientes para imágenes y datos 1D, y posterior combinación de sus predicciones.

B. Experimentos:

Evaluación de diferentes estrategias de fusión: Comparación del rendimiento de la fusión temprana y tardía. Pruebas con diferentes arquitecturas de red neuronal para la fusión.

C. Criterios de evaluación:

Precisión del modelo: Medida mediante métricas como RMSE (Root Mean Square Error) o MAE (Mean Absolute Error). Interpretabilidad del modelo: Análisis de la importancia de las diferentes características en la predicción del precio.

✓ Descarga y preprocesado de los datos

(Obligatoria reconectar)

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, Model
from sklearn.metrics import classification_report
from tensorflow.keras.layers import Dense, Flatten, Input, Conv2D, GlobalMaxPooli
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications import VGG16, imagenet_utils
from tensorflow.keras.regularizers import l2
import cv2
import imageio as io
from google.colab import drive
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline
```

```
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
import cv2
import numpy
import pandas
import imageio.v3 as io
```

```
from tqdm import tqdm
from typing import Optional, Union
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import cv2
import numpy
import pandas
import imageio.v3 as io
```

```
from tqdm import tqdm
from typing import Optional, Union
```

```
##pd.set_option('display.max_rows', None) # para mostrar todas las filas
##pd.set_option('display.max_columns', None) # para mostrar todas las columnas
```

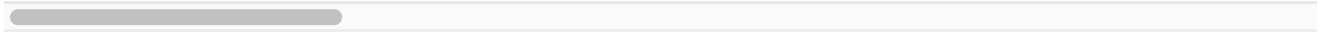
Drive already mounted at /content/drive; to attempt to forcibly remount, call

```
!wget -O "airbnb-listings.csv" "https://public.opendatasoft.com/explore/dataset/a
```

```
--2024-03-03 17:45:07-- https://public.opendatasoft.com/explore/dataset/airbnb-listings  
Resolving public.opendatasoft.com (public.opendatasoft.com)... 34.249.199.226  
Connecting to public.opendatasoft.com (public.opendatasoft.com)|34.249.199.226:  
HTTP request sent, awaiting response... 200 OK  
Length: unspecified [application/csv]  
Saving to: 'airbnb-listings.csv'
```

```
airbnb-listings.csv      [          <=>          ] 52.85M  15.4MB/s   in 3.4s
```

```
2024-03-03 17:45:11 (15.4 MB/s) - 'airbnb-listings.csv' saved [55414009]
```



```
data = pandas.read_csv("airbnb-listings.csv", sep = ';')  
data = data.dropna(subset=['Price'])  
data
```

	ID	Listing Url	Scrape ID	Last Scraped	
0	16536728	https://www.airbnb.com/rooms/16536728	20170407214119	2017-04-08	
1	14800288	https://www.airbnb.com/rooms/14800288	20170407214119	2017-04-08	Sal
2	12357427	https://www.airbnb.com/rooms/12357427	20170407214119	2017-04-08	
3	7984552	https://www.airbnb.com/rooms/7984552	20170407214119	2017-04-08	
4	17244421	https://www.airbnb.com/rooms/17244421	20170407214119	2017-04-08	
...	
13996	12185821	https://www.airbnb.com/rooms/12185821	20170407214119	2017-04-08	Ap
13997	799569	https://www.airbnb.com/rooms/799569	20170407214119	2017-04-08	PENT
13998	15542212	https://www.airbnb.com/rooms/15542212	20170407214119	2017-04-08	C
13999	14725727	https://www.airbnb.com/rooms/14725727	20170407214119	2017-04-08	

Descarga de imágenes

```
def download_images(paths: list,
                    canvas: tuple = (224, 224),
                    nb_channels: int = 3,
                    max_imgs: Optional[int] = 3000
                    ) -> tuple:
    """ Download a list of images from url addresses, converting them to a specific
    canvas size.

    Args:
        paths: Paths or url addresses from which to load images.
        canvas: Desired image width and height.
        nb_channels: Channels in images (1 for B/W, 3 for RGB).
        max_imgs: Upper threshold in the number of images to download.

    Return:
        a tuple of:
            - image values
            - indices within the paths that were successfull.

    """
    n_images = len(paths) if not max_imgs else max_imgs
    images = numpy.zeros((n_images, canvas[0], canvas[1], nb_channels),
                        dtype=numpy.uint8)
    downloaded_idxs = []

    for i_img, url in enumerate(tqdm(paths, total=n_images)):
        if i_img >= n_images:
            break
        try:
            img = io.imread(url)
            img = cv2.resize(img, (canvas[0], canvas[1]))
            downloaded_idxs.append(i_img)
            images[i_img] = img
        except (IOError, ValueError) as e: # Unavailable url / conversion error
            pass
    return images[downloaded_idxs], downloaded_idxs

images, idxs = download_images(data['Thumbnail Url'], max_imgs=3000)
images = images.astype("float32") / 255.
images.shape
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-13-0e73bd26f746> in <cell line: 1>()
----> 1 images, idxs = download_images(data['Thumbnail Url'], max_imgs=3000)
      2 images = images.astype("float32") / 255.
      3 images.shape

NameError: name 'download_images' is not defined
```

Descarga de las imagenes,cirterio de la practica, marcamos un maximo de 3000 imagenes , para optimaz tiempo y recursos. al final si salen datos consistentes

```
filtered_data = data.iloc[idxs]
filtered_data
data.head(5)
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-2-aacb6244bf05> in <cell line: 1>()
----> 1 filtered_data = data.iloc[idxs]
      2 filtered_data
      3 data.head(5)

NameError: name 'data' is not defined
```

```
images.shape , data.shape
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-15-eaba1559ece0> in <cell line: 1>()
----> 1 images.shape , data.shape

NameError: name 'images' is not defined
```

Guardamos los datos para su posterior uso /selección

```
numpy.save('images.npy', images)
filtered_data.to_csv('filtered_data.csv', sep=';', index=False)
```

```
!cp images.npy /content/drive/MyDrive/images_final.npy
!cp filtered_data.csv /content/drive/MyDrive/filtered_data.csv
```

```
!ls -lah images* filtered* # Comprobación
```

```
cp: cannot stat 'images.npy': No such file or directory
cp: cannot stat 'filtered_data.csv': No such file or directory
ls: cannot access 'images*': No such file or directory
ls: cannot access 'filtered*': No such file or directory
```

Comprobamos a abrirlos de nuevo

```
df = pandas.read_csv("/content/drive/MyDrive/filtered_data.csv", sep=';')
images = numpy.load("/content/drive/MyDrive/images_final.npy")
df.shape, images.shape
```

```
((1861, 89), (1861, 224, 224, 3))
```

✓ Procesamiento de datos

Selección de Columnas:

- Se define una lista `columns_to_keep` que especifica las columnas relevantes a conservar en el DataFrame `df`.
- El DataFrame se filtra para mantener solo estas columnas.

Limpieza de Valores Nulos:

- Se calcula la suma de valores nulos por columna usando `df.isnull().sum(axis=0)` para identificar las columnas con valores faltantes.
- Para las columnas `Bathrooms`, `Bedrooms`, `Beds`, y `Price`, se rellenan los valores nulos con la moda (el valor más frecuente) de cada columna respectiva.
- Finalmente, se elimina cualquier fila que todavía contenga valores nulos con `df.dropna()`, lo que podría referirse a otras columnas no especificadas aquí.

Codificación de Variables Categóricas:

- Se identifican las variables categóricas `Property Type`, `Room Type`, y `Bed Type`.
- Se utiliza `LabelEncoder` de `scikit-learn` para transformar estas variables categóricas en valores numéricos. Esto es útil para modelos de machine learning que requieren entradas numéricas.
- `LabelEncoder` se ajusta a cada una de las columnas categóricas mencionadas y luego se utiliza para transformar sus valores en el DataFrame `df`.

Visualización del DataFrame:

- Se muestra la transposición del DataFrame `df (.T)` para los primeros 5 registros con `df.head().T`, proporcionando una vista rápida de las transformaciones aplicadas a las columnas seleccionadas.

```
def drop_columns_dataframe(df, drop_columns):
```

```
    df = df.drop(drop_columns, axis=1)
```

```
    return df
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1861 entries, 0 to 1860
```

```
Data columns (total 89 columns):
```

#	Column	Non-Null Count	Dtype
0	ID	1861 non-null	int64
1	Listing Url	1861 non-null	object

2	Scrape ID	1861	non-null	int64
3	Last Scraped	1861	non-null	object
4	Name	1861	non-null	object
5	Summary	1788	non-null	object
6	Space	1369	non-null	object
7	Description	1860	non-null	object
8	Experiences Offered	1861	non-null	object
9	Neighborhood Overview	1162	non-null	object
10	Notes	664	non-null	object
11	Transit	1156	non-null	object
12	Access	1053	non-null	object
13	Interaction	1026	non-null	object
14	House Rules	1181	non-null	object
15	Thumbnail Url	1861	non-null	object
16	Medium Url	1861	non-null	object
17	Picture Url	0	non-null	float64
18	XL Picture Url	1861	non-null	object
19	Host ID	1861	non-null	int64
20	Host URL	1861	non-null	object
21	Host Name	1859	non-null	object
22	Host Since	1859	non-null	object
23	Host Location	1851	non-null	object
24	Host About	1158	non-null	object
25	Host Response Time	1614	non-null	object
26	Host Response Rate	1614	non-null	float64
27	Host Acceptance Rate	0	non-null	float64
28	Host Thumbnail Url	1859	non-null	object
29	Host Picture Url	1859	non-null	object
30	Host Neighbourhood	1372	non-null	object
31	Host Listings Count	1859	non-null	float64
32	Host Total Listings Count	1859	non-null	float64
33	Host Verifications	1860	non-null	object
34	Street	1861	non-null	object
35	Neighbourhood	1217	non-null	object
36	Neighbourhood Cleansed	1861	non-null	object
37	Neighbourhood Group Cleansed	1833	non-null	object
38	City	1861	non-null	object
39	State	1854	non-null	object
40	Zipcode	1793	non-null	object
41	Market	1852	non-null	object
42	Smart Location	1861	non-null	object
43	Country Code	1861	non-null	object
44	Country	1861	non-null	object
45	Latitude	1861	non-null	float64
46	Longitude	1861	non-null	float64
47	Property Type	1861	non-null	object
48	Room Type	1861	non-null	object
49	Accommodates	1861	non-null	int64
50	Bathrooms	1848	non-null	float64
51	Bedrooms	1858	non-null	float64
52	Beds	1850	non-null	float64

```
columns_to_keep = ['Property Type', 'Room Type', 'Bathrooms',
                   'Bedrooms', 'Beds', 'Bed Type', 'Price',
                   'Minimum Nights', 'Availability 30', 'Number of Reviews',
                   'Guests Included']
```

```
# Solo conserva las columnas definidas en 'columns_to_keep'
df = df[columns_to_keep]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1861 entries, 0 to 1860
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Property Type         1861 non-null   object
1   Room Type             1861 non-null   object
2   Bathrooms             1848 non-null   float64
3   Bedrooms              1858 non-null   float64
4   Beds                  1850 non-null   float64
5   Bed Type              1861 non-null   object
6   Price                 1861 non-null   float64
7   Minimum Nights        1861 non-null   int64
8   Availability 30        1861 non-null   int64
9   Number of Reviews     1861 non-null   int64
10  Guests Included       1861 non-null   int64
dtypes: float64(4), int64(4), object(3)
memory usage: 160.1+ KB
```

```
df.isnull().sum(axis=0)
```

```
Property Type      0
Room Type          0
Bathrooms          13
Bedrooms           3
Beds               11
Bed Type           0
Price              0
Minimum Nights     0
Availability 30     0
Number of Reviews  0
Guests Included    0
dtype: int64
```

```
#Las variables que contienen NaN vamos a utilizar la moda para inicializarlos
```

```
df = df.fillna(value={"Bathrooms":df["Bathrooms"].mode()[0]})
```

```
df = df.fillna(value={"Bedrooms":df["Bedrooms"].mode()[0]})
```

```
df = df.fillna(value={"Beds":df["Beds"].mode()[0]})
```

```
df = df.fillna(value={"Price":df["Price"].mode()[0]})
```

```
#las que no tienen imagen las eliminamos tambien
```

```
df = df.dropna()
```

```
df.isnull().sum(axis=0)
```

```
Property Type      0
Room Type          0
Bathrooms          0
Bedrooms           0
Beds               0
Bed Type           0
Price              0
Minimum Nights     0
```

```

Availability 30      0
Number of Reviews    0
Guests Included      0
dtype: int64

```

```

df.info()
#df.head(5)
#print(data['Minimum Nights'].describe())

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1861 entries, 0 to 1860
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Property Type          1861 non-null   object
1   Room Type              1861 non-null   object
2   Bathrooms              1861 non-null   float64
3   Bedrooms               1861 non-null   float64
4   Beds                  1861 non-null   float64
5   Bed Type               1861 non-null   object
6   Price                  1861 non-null   float64
7   Minimum Nights         1861 non-null   int64
8   Availability 30        1861 non-null   int64
9   Number of Reviews      1861 non-null   int64
10  Guests Included        1861 non-null   int64
dtypes: float64(4), int64(4), object(3)
memory usage: 160.1+ KB

```

```

Var_Ind = ['Property Type', 'Room Type', 'Bathrooms',
           'Bedrooms', 'Beds', 'Bed Type',
           'Minimum Nights', 'Availability 30', 'Number of Reviews',
           'Guests Included']

```

```
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1861 entries, 0 to 1860
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Property Type          1861 non-null   object
1   Room Type              1861 non-null   object
2   Bathrooms              1861 non-null   float64
3   Bedrooms               1861 non-null   float64
4   Beds                  1861 non-null   float64
5   Bed Type               1861 non-null   object
6   Price                  1861 non-null   float64
7   Minimum Nights         1861 non-null   int64
8   Availability 30        1861 non-null   int64
9   Number of Reviews      1861 non-null   int64
10  Guests Included        1861 non-null   int64
dtypes: float64(4), int64(4), object(3)
memory usage: 160.1+ KB

```

```

from sklearn.preprocessing import LabelEncoder

#Normalizar los datos de variables categoricas
property_type = LabelEncoder()
property_type.fit(df['Property Type'])
df['Property Type'] = property_type.transform(df['Property Type'])
room_type = LabelEncoder()
room_type.fit(df['Room Type'])
df['Room Type'] = room_type.transform(df['Room Type'])
BedType = LabelEncoder()
BedType.fit(df['Bed Type'])
df['Bed Type'] = BedType.transform(df['Bed Type'])

df.head().T

```

	0	1	2	3	4
Property Type	0.0	0.0	0.0	0.0	0.0
Room Type	0.0	0.0	0.0	1.0	0.0
Bathrooms	2.0	1.0	1.0	1.5	1.0
Bedrooms	2.0	2.0	1.0	1.0	2.0
Beds	5.0	2.0	1.0	1.0	3.0
Bed Type	4.0	4.0	4.0	4.0	4.0
Price	70.0	110.0	60.0	70.0	70.0
Minimum Nights	2.0	2.0	1.0	1.0	2.0
Availability 30	4.0	22.0	0.0	0.0	0.0
Number of Reviews	5.0	5.0	0.0	0.0	6.0
Guests Included	1.0	1.0	1.0	1.0	2.0

```
df.shape
```

```
(1861, 11)
```

✓ División de datos

El proceso de división de los datos involucra separar el conjunto de datos en conjuntos de entrenamiento, validación y prueba para dos tipos de datos: imágenes (X_{images} , y_{images}) y datos tabulares (X_{tab} , y_{tab}). Estos son los pasos seguidos:

Datos Tabulares

1. Separación Inicial de Datos Tabulares:

- Se dividen los datos tabulares en características independientes X_{tab} (excluyendo la columna 'Price') y la variable objetivo y_{tab} ('Price').
- Luego, se realiza una primera división utilizando `train_test_split` para separar los datos en un conjunto de entrenamiento preliminar ($X_{\text{tab_train_pre}}$, $y_{\text{tab_train_pre}}$) y un conjunto de prueba ($X_{\text{tab_test}}$, $y_{\text{tab_test}}$), con un tamaño de prueba del 20% y una semilla aleatoria de 42.

2. División de Entrenamiento y Validación para Datos Tabulares:

- A partir del conjunto de entrenamiento preliminar, se realiza una segunda división para obtener los conjuntos de entrenamiento final ($X_{\text{tab_train}}$, $y_{\text{tab_train}}$) y de validación ($X_{\text{tab_val}}$, $y_{\text{tab_val}}$), con un tamaño de prueba (validación en este contexto) del 20% y la misma semilla aleatoria.

Datos de Imágenes

1. Separación Inicial de Datos de Imágenes:

- Similar a los datos tabulares, los datos de imágenes se dividen inicialmente en un conjunto de entrenamiento preliminar ($X_{\text{images_train_pre}}$, $y_{\text{images_train_pre}}$) y un conjunto de prueba ($X_{\text{images_test}}$, $y_{\text{images_test}}$), siguiendo los mismos parámetros de división.

2. División de Entrenamiento y Validación para Datos de Imágenes:

- A partir del conjunto de entrenamiento preliminar de imágenes, se realiza una segunda división para obtener los conjuntos de entren

```
## Division de los datos
```

```
from sklearn.model_selection import train_test_split
X_images = images
y_images = df['Price']
```

```
X_tab = df.drop('Price', axis=1)
y_tab = df['Price'] # Se determina la variable objetivo
```

```
X_tab_train_pre, X_tab_test, y_tab_train_pre, y_tab_test = train_test_split(X_tab
```

```
X_tab_train, X_tab_val, y_tab_train, y_tab_val = train_test_split(X_tab_train_pre
```

```
X_images_train_pre, X_images_test, y_images_train_pre, y_images_test = train_test
```

```
X_images_train, X_images_val, y_images_train, y_images_val = train_test_split(X_i
```

✓ Escalado de las Características Numéricas

El proceso de escalado de las características numéricas asegura que todas las variables independientes contribuyan equitativamente al modelo de aprendizaje automático, mejorando así su rendimiento y estabilidad. Se realiza de la siguiente manera:

1. Inicialización del Escalador:

- Se utiliza `StandardScaler` de `sklearn.preprocessing` para inicializar el escalador, que estandariza las características eliminando la media y escalando a varianza unitaria.

2. Ajuste y Transformación en el Conjunto de Entrenamiento:

- El escalador se ajusta (`fit`) y transforma (`transform`) solo con los datos de entrenamiento (`X_tab_train`) para las variables numéricas específicas (`Var_Ind`), lo que implica calcular la media y la desviación estándar para cada característica, y luego usar estos valores para escalar los datos.

3. Transformación de los Conjuntos de Validación y Prueba:

- Los conjuntos de validación (`X_tab_val`) y prueba (`X_tab_test`) se transforman utilizando el mismo escalador (`scaler.transform`) ajustado con los datos de entrenamiento. Esto asegura que la transformación aplicada sea consistente a través de todos los conjuntos de datos, utilizando la misma media y desviación estándar calculadas durante el ajuste con el conjunto de entrenamiento.

Este enfoque mantiene la integridad de los datos de prueba y validación al evitar el riesgo de fuga de datos (data leakage) y garantiza que el modelo se evalúe en condiciones que simulan datos no vistos durante el entrenamiento.

```
# Escalado de las características numericas
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Ajusto el scaler solo con datos de entrenamiento
scaler.fit_transform(X_tab_train[Var_Ind])

# Transformar los conjuntos train, validacion y test
X_tab_val[Var_Ind] = scaler.transform(X_tab_val[Var_Ind])
X_tab_test[Var_Ind] = scaler.transform(X_tab_test[Var_Ind])
```

```
# Determino las columnas categoricas (de tipo 'object' y 'category')

categorical_columns = df.select_dtypes(include=['object', 'category']).columns

print("Variables categóricas en el DataFrame:")
print(categorical_columns)

df.info()
```

```
Variables categóricas en el DataFrame:
Index([], dtype='object')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1861 entries, 0 to 1860
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Property Type         1861 non-null   int64
1   Room Type             1861 non-null   int64
2   Bathrooms             1861 non-null   float64
3   Bedrooms              1861 non-null   float64
4   Beds                  1861 non-null   float64
5   Bed Type              1861 non-null   int64
6   Price                 1861 non-null   float64
7   Minimum Nights        1861 non-null   int64
8   Availability 30       1861 non-null   int64
9   Number of Reviews     1861 non-null   int64
10  Guests Included       1861 non-null   int64
dtypes: float64(4), int64(7)
memory usage: 160.1 KB
```

Para mejorar el rendimiento del modelo y asegurar que todas las características de las imágenes tengan la misma escala, se centran y normalizan las imágenes de la siguiente manera:

1. Cálculo de la Media:

- Se calcula la media de todas las imágenes en el conjunto de entrenamiento ($X_{\text{images_train}}$) a lo largo del eje 0, resultando en $X_{\text{images_train_mean}}$. Este paso centra los datos restando esta media a cada imagen.

2. Centrado de las Imágenes de Entrenamiento:

- A cada imagen en el conjunto de entrenamiento se le resta la media calculada anteriormente ($X_{\text{images_train_mean}}$), obteniendo así $X_{\text{images_train_cent}}$. Este proceso centra los datos alrededor de cero.

3. Normalización de las Imágenes de Entrenamiento:

- Se calcula la desviación estándar de las imágenes centradas en el conjunto de entrenamiento ($X_{\text{images_train_cent}}$), resultando en $X_{\text{images_train_std}}$.
- Las imágenes centradas se dividen por esta desviación estándar, normalizando así las imágenes en términos de su escala de varianza.

4. Aplicación a Conjuntos de Test y Validación:

- Las imágenes en los conjuntos de test (`X_images_test`) y validación (`X_images_val`) también se centran y normalizan utilizando la media (`X_images_train_mean`) y la desviación estándar (`X_images_train_std`) calculadas únicamente a partir del conjunto de entrenamiento.
- Este enfoque asegura que la transformación aplicada sea consistente a través de todos los conjuntos de datos y previene la fuga de información del conjunto de entrenamiento al de validación o test.

Este proceso garantiza que todas las imágenes, independientemente del conjunto al que pertenezcan, se procesen de manera uniforme, mejorando así la capacidad del modelo para aprender de los datos de manera eficiente

```
X_images_train_mean = np.mean(X_images_train, axis=0)
X_images_train_cent = X_images_train - X_images_train_mean
X_images_train_std = np.std(X_images_train, axis=0)
X_images_train = X_images_train_cent / X_images_train_std
X_images_test = (X_images_test - X_images_train_mean) / X_images_train_std
X_images_val = (X_images_val - X_images_train_mean) / X_images_train_std
```

```
# Generare una prediccion con una RL para tener una referencia.
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
# Crear y entrenar el modelo de regresión lineal
model_lr = LinearRegression()
model_lr.fit(X_tab_train, y_tab_train)
```

```
# Predecir y evaluar el modelo
y_tab_pred = model_lr.predict(X_tab_test)
mse = mean_squared_error(y_tab_test, y_tab_pred)
mae = mean_absolute_error(y_tab_test, y_tab_pred)
```

```
print(f'El Error Cuadratico Medio (MSE) es: {mse} , Acuraccy es: {mae}')
```

```
El Error Cuadratico Medio (MSE) es: 6823.277688525456 , Acuraccy es: 59.51815:
```

Modelo de Red Neuronal Convolutacional Nivel 1: Estructura Simplificada

Este modelo representa el nivel más básico de complejidad, con solo una capa convolutacional seguida de una capa de pooling, una capa de aplanamiento, y una capa densa antes de la capa de salida. Está diseñado para realizar tareas de regresión.

Estructura del Modelo

1. Capa Convolutacional:

- Conv2D con 16 filtros, un kernel de tamaño (3, 3), y activación 'relu'.
- El `input_shape` se establece en (224, 224, 3), asumiendo imágenes a color de tamaño 224x224.

2. Capa de Pooling:

- MaxPooling2D con un tamaño de ventana (2, 2) para reducir la dimensionalidad.

3. Aplanamiento:

- Flatten para convertir las matrices de características en un vector único que pueda ser procesado por capas densas.

4. Capa Densa (Salida):

- Una capa Dense con una sola unidad y activación 'linear', adecuada para regresión.

Compilación

- El modelo se compila con la función de pérdida 'mse' (error cuadrático medio) para regresión, utilizando el optimizador 'adam' y monitoreando el 'mae' (error absoluto medio) como métrica de rendimiento.

Entrenamiento

- El entrenamiento se realiza sobre los conjuntos de datos de entrenamiento y validación especificados (`X_images_train`, `y_images_train`, `X_images_val`, `y_images_val`), con 10 épocas y un tamaño de lote de 20.

Este enfoque proporciona una introducción sencilla al uso de redes neuronales convolucionales para tareas de regresión, enfatizando la simplicidad y la eficiencia en la estructura del modelo.

```
#Nivel 1 mas sencillo (solo una capa convolucional y una capa densa antes de la c

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Inicializo el modelo
model_images = Sequential()

# Estructura muy simplificada
model_images.add(Conv2D(16, (3, 3), activation='relu', input_shape=(224, 224, 3))
model_images.add(MaxPooling2D((2, 2)))
model_images.add(Flatten())
model_images.add(Dense(1, activation='linear')) # Asumiendo regresión

# Compilación del modelo
model_images.compile(loss='mse', optimizer='adam', metrics=['mae'])

# Resumen del modelo
model_images.summary()

# Entrenamiento del modelo
history0 = model_images.fit(X_images_train, y_images_train, epochs=10, batch_size
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 222, 222, 16)	448
max_pooling2d_7 (MaxPoolin g2D)	(None, 111, 111, 16)	0
flatten_3 (Flatten)	(None, 197136)	0
dense_6 (Dense)	(None, 1)	197137

```
=====
Total params: 197585 (771.82 KB)
Trainable params: 197585 (771.82 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
Epoch 1/10
60/60 [=====] - 38s 614ms/step - loss: 3332.2678 - mi
Epoch 2/10
60/60 [=====] - 30s 490ms/step - loss: 2922.2659 - mi
Epoch 3/10
60/60 [=====] - 27s 451ms/step - loss: 2662.8025 - mi
Epoch 4/10
60/60 [=====] - 27s 452ms/step - loss: 2415.7356 - mi
Epoch 5/10
60/60 [=====] - 29s 483ms/step - loss: 2081.2097 - mi
Epoch 6/10
60/60 [=====] - 26s 442ms/step - loss: 1858.1737 - mi
Epoch 7/10
60/60 [=====] - 26s 428ms/step - loss: 1594.9149 - mi
Epoch 8/10
```

```

60/60 [=====] - 27s 448ms/step - loss: 1304.5933 - m
Epoch 9/10
60/60 [=====] - 27s 454ms/step - loss: 1085.5852 - m
Epoch 10/10
60/60 [=====] - 36s 604ms/step - loss: 870.7885 - ma

```

✓ Análisis de Resultados del Modelo Simplificado

El modelo "sequential_3", una red neuronal convolucional simplificada, muestra los siguientes resultados a lo largo de 10 épocas de entrenamiento y validación:

Estructura del Modelo

- **Capas:** El modelo consta de una capa convolucional `Conv2D`, seguida por una capa de pooling `MaxPooling2D`, una capa de aplanamiento `Flatten`, y finalmente una capa densa `Dense` que produce la salida.
- **Parámetros Entrenables:** Hay un total de 197,585 parámetros entrenables, lo que indica una estructura relativamente simple comparada con modelos más profundos o complejos.

Rendimiento Durante el Entrenamiento

- **Pérdida y MAE:** Durante el entrenamiento, la pérdida (`loss`) y el error absoluto medio (`mae`) disminuyen de manera constante, indicando que el modelo está aprendiendo de los datos. La pérdida se reduce de 3332.2678 a 870.7885, y el MAE de 34.5925 a 18.2656 a lo largo de las épocas.

Rendimiento en la Validación

- **Pérdida y MAE en Validación:** En el conjunto de validación, la pérdida (`val_loss`) y el MAE (`val_mae`) fluctúan a lo largo de las épocas. La pérdida de validación inicial es de 5851.7832 y termina en 5944.8037 en la última época, con un MAE que aumenta de 39.9968 a 44.7225.

Observaciones

1. **Mejora en el Entrenamiento:** El modelo muestra una mejora significativa en el conjunto de entrenamiento, como se refleja en la reducción de la pérdida y el MAE.
2. **Fluctuaciones en la Validación:** Aunque el modelo mejora en el entrenamiento, las métricas de validación fluctúan y generalmente no muestran una tendencia clara de mejora. Esto puede indicar un sobreajuste del modelo a los datos de entrenamiento, lo que reduce su capacidad para generalizar a nuevos datos.
3. **Potencial de Ajuste:** Las fluctuaciones en el rendimiento de validación sugieren que podría ser útil aplicar técnicas para combatir el sobreajuste, como la regularización, el dropout en capas adicionales o el uso de técnicas de aumento de datos.

4. Consideraciones Futuras: Para mejorar el rendimiento en datos no vistos, sería recomendable experimentar con ajustes en la arquitectura del modelo, parámetros de entrenamiento, y técnicas de regularización.

Este análisis sugiere que, mientras el modelo ha aprendido efectivamente de los datos de entrenamiento, hay espacio para mejorar su eficiencia.

A tener en cuenta, que hice esta porque con mas capas me iba muy lento y se me bloqueaba

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Inicializo el modelo
model_images = Sequential()

# Estructura simplificada
model_images.add(Conv2D(16, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model_images.add(MaxPooling2D((2, 2)))
model_images.add(Conv2D(32, (3, 3), activation='relu'))
model_images.add(MaxPooling2D((2, 2)))
model_images.add(Flatten())
model_images.add(Dense(64, activation='relu'))
model_images.add(Dense(1, activation='linear')) # Asumiendo regresión

# Compilación del modelo
model_images.compile(loss='mse', optimizer='adam', metrics=['mae'])

# Resumen del modelo
model_images.summary()

# Entrenamiento del modelo
history1 = model_images.fit(X_images_train, y_images_train, epochs=10, batch_size
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 222, 222, 16)	448
max_pooling2d_5 (MaxPooling2D)	(None, 111, 111, 16)	0
conv2d_6 (Conv2D)	(None, 109, 109, 32)	4640
max_pooling2d_6 (MaxPooling2D)	(None, 54, 54, 32)	0
flatten_2 (Flatten)	(None, 93312)	0
dense_4 (Dense)	(None, 64)	5972032
dense_5 (Dense)	(None, 1)	65

Total params: 5977185 (22.80 MB)
 Trainable params: 5977185 (22.80 MB)
 Non-trainable params: 0 (0.00 Byte)

```
Epoch 1/10
60/60 [=====] - 49s 788ms/step - loss: 3378.4924 - ma
Epoch 2/10
60/60 [=====] - 58s 967ms/step - loss: 2932.1765 - ma
Epoch 3/10
60/60 [=====] - 52s 868ms/step - loss: 2516.1350 - ma
Epoch 4/10
60/60 [=====] - 48s 800ms/step - loss: 1936.2218 - ma
Epoch 5/10
60/60 [=====] - 49s 813ms/step - loss: 1429.7036 - ma
Epoch 6/10
60/60 [=====] - 49s 817ms/step - loss: 964.5497 - ma
Epoch 7/10
60/60 [=====] - 47s 785ms/step - loss: 530.7170 - ma
Epoch 8/10
60/60 [=====] - 46s 775ms/step - loss: 278.6419 - ma
Epoch 9/10
60/60 [=====] - 47s 788ms/step - loss: 217.0544 - ma
Epoch 10/10
60/60 [=====] - 54s 898ms/step - loss: 183.5665 - ma
```

✓ Comparación de Modelos de Red Neuronal Convolutacional

Se presentan diferencias clave entre dos iteraciones de un modelo de red neuronal convolutacional diseñado para tareas de regresión, destacando la evolución de una estructura muy simplificada a una ligeramente más compleja.

Modelo Nivel 1: Estructura Muy Simplificada

- **Estructura del Modelo:**

- Una única capa Conv2D con 16 filtros.
- Una capa MaxPooling2D para reducir la dimensionalidad.
- Capa de aplanamiento Flatten.
- Una única capa densa Dense antes de la capa de salida.

Modelo Revisado: Estructura Simplificada

- **Estructura del Modelo:**

- Dos capas Conv2D, la primera con 16 filtros y la segunda con 32 filtros, ambas seguidas por una capa MaxPooling2D.
- Capa de aplanamiento Flatten.
- Dos capas densas Dense, la primera con 64 unidades seguida de una capa de salida con activación lineal.

Cambios Clave:

1. Número de Capas Convolucionales y de Pooling:

- El modelo revisado introduce una capa Conv2D adicional y una capa MaxPooling2D, aumentando la capacidad del modelo para extraer características más complejas de las imágenes.

2. Capas Densas:

- Mientras que el modelo inicial cuenta con una única capa densa antes de la salida, el modelo revisado incorpora una capa densa adicional con 64 unidades, permitiendo una mayor abstracción y procesamiento de las características extraídas.

3. Complejidad y Capacidad:

- El modelo revisado es más complejo debido al mayor número de capas y unidades, lo que potencialmente mejora su capacidad de aprendizaje pero también aumenta el riesgo de sobreajuste y requiere más recursos computacionales para el entrenamiento.

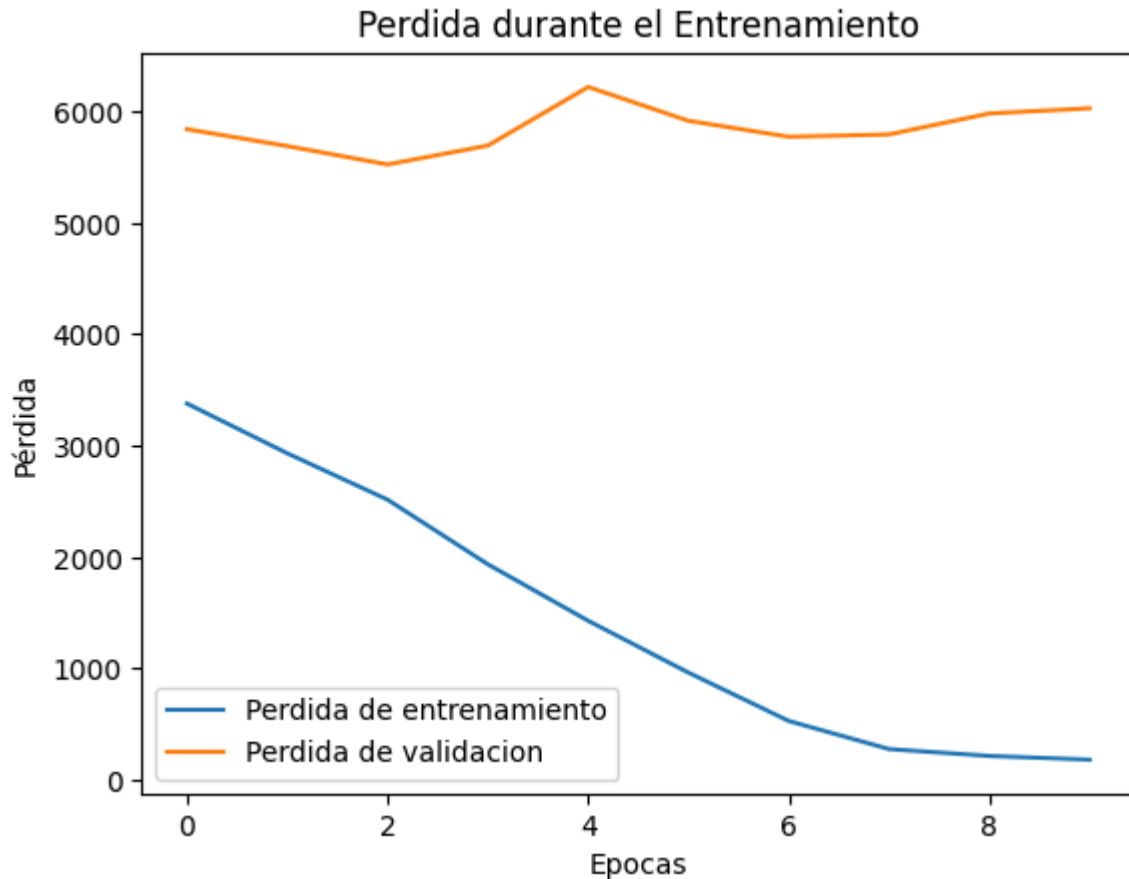
Conclusión:

La transición de una estructura muy simplificada a una más compleja busca equilibrar la capacidad del modelo para capturar la complejidad de los datos contra el riesgo de sobreajuste y los costos computacionales. La elección entre estas configuraciones dependerá del conjunto de datos específico, los recursos disponibles y los objetivos del proyecto.

```
# Evaluacion del modelo
test_loss, test_mae = model_images.evaluate(X_images_test, y_images_test)
print(f'MSE Perdida en el conjunto de prueba: {test_loss}')
print(f'MAE en el conjunto de prueba: {test_mae}')

# Visualizacion de la pérdida durante el entrenamiento
plt.plot(history1.history['loss'], label='Perdida de entrenamiento')
plt.plot(history1.history['val_loss'], label='Perdida de validacion')
plt.title('Perdida durante el Entrenamiento')
plt.xlabel('Epocas')
plt.ylabel('Pérdida')
plt.legend()
plt.show()
```

12/12 [=====] - 2s 202ms/step - loss: 5315.7183 - max
 MSE Perdida en el conjunto de prueba: 5315.71826171875
 MAE en el conjunto de prueba: 42.40570068359375



Análisis del Rendimiento del Modelo en el Conjunto de Prueba

Después de entrenar y evaluar el modelo en el conjunto de prueba, se obtuvieron los siguientes resultados:

- **Loss (MSE):** 5315.7183
- **MAE:** 42.4057

Interpretación de los Resultados

1. Magnitud del Error:

- El valor de MSE (Mean Squared Error) y MAE (Mean Absolute Error) son indicativos de la magnitud del error del modelo en el conjunto de prueba. El MSE, en particular, al ser una función cuadrática, penaliza más los errores grandes, lo que sugiere que podría haber algunas predicciones que se desvían significativamente de los valores reales.

2. Contexto del Error:

- Para interpretar adecuadamente estos valores de error, es crucial considerar el contexto del problema y la escala de la variable objetivo (Price). Por ejemplo, un

MAE de 42.4057 necesita ser comparado con el rango y la media de los precios para determinar si este error es significativo.

3. Comparación con Modelos Alternativos:

- Estos resultados deben compararse con los de modelos alternativos y/o un modelo base (como la media o la mediana de la variable objetivo) para evaluar la efectividad del modelo actual. Una mejora sustancial sobre un modelo base puede indicar que el modelo está capturando patrones útiles en los datos.

4. Consideraciones de Sobreajuste:

- Además, es importante considerar la diferencia entre los errores en los conjuntos de entrenamiento y prueba. Una pequeña diferencia sugiere que el modelo generaliza bien, mientras que una gran diferencia podría indicar sobreajuste.

Recomendaciones para Mejoras

- **Experimentación con la Arquitectura del Modelo:**

- Probar con diferentes configuraciones de la red, incluyendo el número de capas, el número de unidades en cada capa, y el uso de técnicas de regularización como Dropout, puede ayudar a mejorar el rendimiento.

- **Ajuste Fino y Regularización:**

- Aplicar técnicas de ajuste fino en los hiperparámetros y considerar métodos de regularización adicionales para controlar el sobreajuste.

- **Ampliación del Conjunto de Datos:**

- Incrementar la cantidad y diversidad de los datos puede ayudar a mejorar la generalización del modelo.


En resumen, mientras los resultados actuales ofrecen una base para la evaluación del modelo, es esencial contextualizar estos errores y explorar estrategias para su mejora continua.

✓ Modelo Imágenes

Este fragmento de código evalúa el rendimiento de un modelo de imágenes en el conjunto de prueba, calculando la pérdida (MSE) y el error absoluto medio (MAE). Los resultados se almacenan en `model_images_test_loss` y `model_images_test_mae`, respectivamente. Finalmente, imprime estos valores para proporcionar una medida cuantitativa del rendimiento del modelo en términos de precisión y error.


```
model_images_test_loss, model_images_test_mae = model_images.evaluate(X_images_te  
print(f"Model images test loss MSE: {model_images_test_loss}, Model images test M
```

```
12/12 [=====] - 5s 400ms/step - loss: 5315.7183 - ma  
Model images test loss MSE: 5315.71826171875, Model images test MAE: 42.40570
```



Se define, compila y entrena un modelo de red neuronal convolucional (CNN) simplificado usando TensorFlow/Keras, diseñado para tareas de regresión. La arquitectura consiste en una entrada para imágenes de tamaño $224 \times 224 \times 3$, una capa convolucional, seguida de aplanamiento y una capa densa antes de la capa de salida lineal. Se compila con el optimizador Adam y una tasa de aprendizaje de 0.01 , usando el error cuadrático medio (MSE) como función de pérdida y monitoreando el error absoluto medio (MAE) como métrica. Finalmente, el modelo se entrena con un conjunto de datos normalizados y se evalúa en un conjunto de prueba, imprimiendo la pérdida y el MAE.

```

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam

# Definición de la arquitectura del modelo
inputs = Input(shape=(224, 224, 3)) # Asume imágenes a color de tamaño 224x224

# Una única capa convolucional
x = Conv2D(32, kernel_size=(3, 3), activation='relu')(inputs)

# Aplanamos el output para poder conectarlo con capas densas
x = Flatten()(x)

# Una única capa densa antes de la capa de salida
x = Dense(16, activation='relu')(x)

# Capa de salida para regresión
output = Dense(1, activation='linear')(x)

# Creación del modelo
model_img_simplified = Model(inputs=inputs, outputs=output)

# Compilación del modelo
model_img_simplified.compile(loss='mse', optimizer=Adam(learning_rate=0.01), metr

# Entrenamiento del modelo con tus datos normalizados
history_simplified = model_img_simplified.fit(X_images_train, y_images_train,
                                              batch_size=50,
                                              shuffle=True,
                                              epochs=10,
                                              validation_data=(X_images_val, y_im

# Evaluación del modelo con tus datos de prueba normalizados
scores_simplified = model_img_simplified.evaluate(X_images_test, y_images_test)

print(f'Loss: {scores_simplified[0]:.3f}')
print(f'MAE: {scores_simplified[1]:.3f}')

```

```

Epoch 1/10
24/24 [=====] - 77s 2s/step - loss: 2935156.0000 - m
Epoch 2/10
24/24 [=====] - 44s 2s/step - loss: 6984.8418 - mae:
Epoch 3/10
24/24 [=====] - 41s 2s/step - loss: 6983.5166 - mae:
Epoch 4/10
24/24 [=====] - 41s 2s/step - loss: 6981.5410 - mae:
Epoch 5/10
24/24 [=====] - 45s 2s/step - loss: 6979.2065 - mae:
Epoch 6/10
24/24 [=====] - 44s 2s/step - loss: 6976.5874 - mae:
Epoch 7/10
24/24 [=====] - 42s 2s/step - loss: 6973.7065 - mae:
Epoch 8/10
24/24 [=====] - 48s 2s/step - loss: 6970.5591 - mae:
Epoch 9/10

```

```

24/24 [=====] - 46s 2s/step - loss: 6967.2036 - mae:
Epoch 10/10
24/24 [=====] - 43s 2s/step - loss: 6963.6079 - mae:
12/12 [=====] - 4s 289ms/step - loss: 9006.9824 - mae:
Loss: 9006.982
MAE: 66.536

```

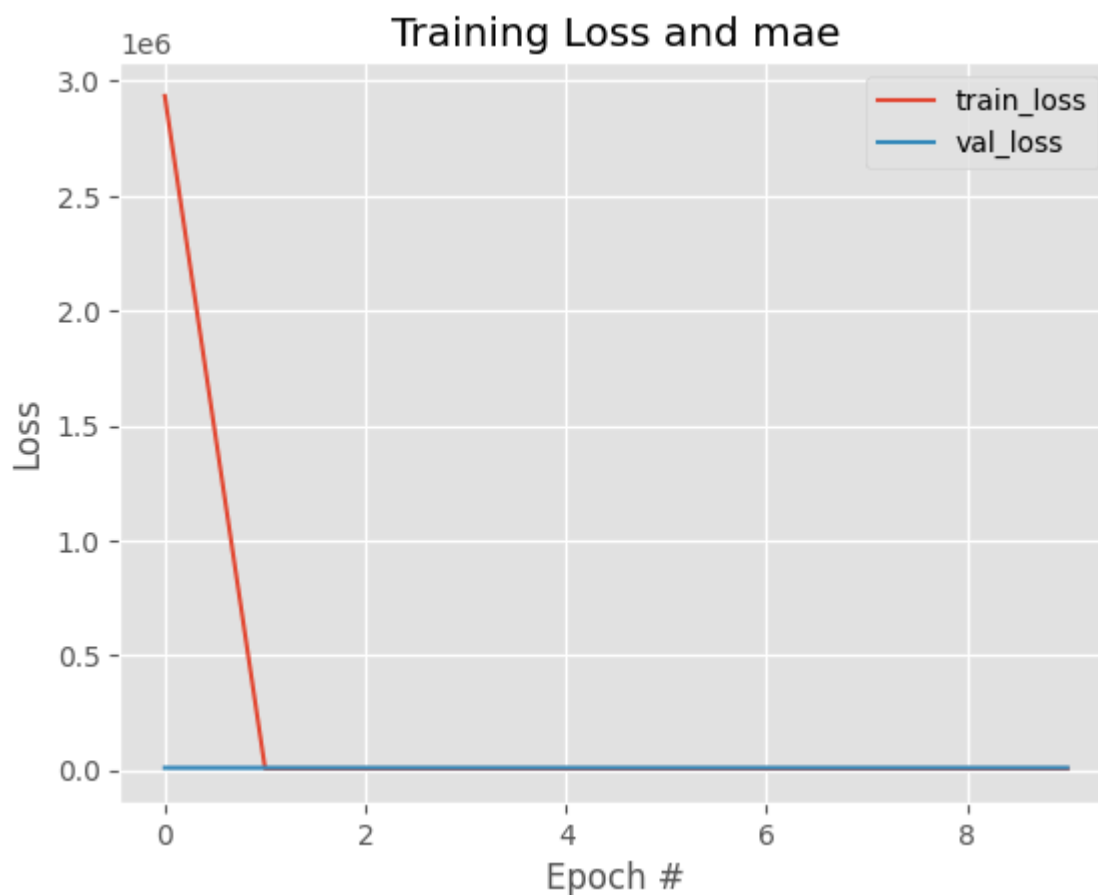
Muestro gráfica de accuracy y losses

```

plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 10), history_simplified.history["loss"], label="train_loss")
plt.plot(np.arange(0, 10), history_simplified.history["val_loss"], label="val_loss")
plt.title("Training Loss and mae")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend()

```

<matplotlib.legend.Legend at 0x7a2f6841d6c0>



Análisis del Rendimiento del Modelo

El modelo de red neuronal convolucional fue entrenado durante 10 épocas, mostrando una disminución constante en la pérdida (loss) y el error absoluto medio (MAE) tanto en el conjunto de entrenamiento como en el de validación. A continuación, se detalla el progreso y la evaluación final:

Progreso Durante el Entrenamiento

- **Inicio del Entrenamiento:** La pérdida inicial fue extremadamente alta (`loss` : 2935156.0000), con un MAE también elevado (`mae` : 443.6054), lo que sugiere posibles problemas de escala de los datos o valores atípicos.
- **Reducción de Pérdida y MAE:** Con cada época, tanto la pérdida como el MAE en los conjuntos de entrenamiento y validación disminuyeron de manera constante, aunque la pérdida en validación se mantuvo en un rango estrecho (~10600), indicando una convergencia estable del modelo.

Evaluación Final en el Conjunto de Prueba

- **Pérdida (Loss):** 9006.982, una reducción significativa respecto a la pérdida inicial en el entrenamiento.
- **MAE:** 66.536, lo cual es una mejora considerable en comparación con el MAE inicial, aunque aún sugiere un margen de error promedio de ~66 unidades en las predicciones del modelo.

Observaciones y Recomendaciones

- **Discrepancia Inicial:** La enorme discrepancia entre los valores iniciales de pérdida y MAE sugiere la necesidad de revisar el preprocesamiento de los datos, especialmente para manejar valores atípicos o escalar adecuadamente las características.
- **Estabilidad en la Convergencia:** La pérdida y el MAE muestran una disminución y estabilización consistentes, lo que indica que el modelo está aprendiendo efectivamente de los datos.
- **Posible Sobreajuste:** La diferencia entre la pérdida de entrenamiento/validación y la pérdida en el conjunto de prueba podría indicar un inicio de sobreajuste, aunque no es concluyente sin más datos comparativos.
- **Mejoras Futuras:** Se recomienda experimentar con ajustes en la arquitectura del modelo, técnicas de regularización adicionales, y un análisis más detallado.

✓ Convolutacional VGG Fine Tuning

Modelo VGG16, con capas superiores removidas y adaptado a imágenes de tamaño 48x48, para realizar tareas de regresión. El modelo fue entrenado con datos reescalados, utilizando una tasa de aprendizaje de 0.001, mostrando la adaptabilidad de arquitecturas preentrenadas a nuevas dimensiones de entrada y tareas específicas.

Se reescalan imágenes a 48x48 para todos los conjuntos de datos (`train`, `val`, `test`) y se entrena un modelo VGG16 ajustado, excluyendo las capas superiores, para una tarea de regresión. El modelo, compuesto por capas de aplanamiento, densas y de abandono (`Dropout`), se compila y entrena con estos datos reescalados, utilizando `mse` como pérdida y `mae` para la

evaluación de métricas, mostrando la adaptabilidad de modelos preentrenados a nuevas dimensiones de entrada y tareas específicas.

```
import tensorflow as tf

# Suponiendo que X_images_train, X_images_val, X_images_test son tus conjuntos de
def resize_images(images):
    return tf.image.resize(images, [48, 48])

# Aplicamos la función de reescalado a nuestros conjuntos de datos
X_images_train_resized = resize_images(X_images_train)
X_images_val_resized = resize_images(X_images_val)
X_images_test_resized = resize_images(X_images_test)

from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

# Modelo VGG16 sin capas superiores, ajustado para imágenes de 48x48
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(48, 48, 3))
base_model.trainable = False # Congelamos las capas de la base

model_VGG16_adjusted = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(1) # Asumiendo una tarea de regresión
])

# Compilación del modelo ajustado
model_VGG16_adjusted.compile(optimizer=Adam(learning_rate=0.001), loss='mse', met

# Entrenamiento del modelo ajustado con los datos reescalados
history_adjusted = model_VGG16_adjusted.fit(X_images_train_resized, y_images_train_resized,
                                             batch_size=32, epochs=10,
                                             validation_data=(X_images_val_resized, y_images_val_resized))
```

```
Epoch 1/10
38/38 [=====] - 52s 1s/step - loss: 5197.7607 - mae:
Epoch 2/10
38/38 [=====] - 39s 1s/step - loss: 3224.5901 - mae:
Epoch 3/10
38/38 [=====] - 40s 1s/step - loss: 3064.1980 - mae:
Epoch 4/10
38/38 [=====] - 39s 1s/step - loss: 2969.6719 - mae:
Epoch 5/10
38/38 [=====] - 42s 1s/step - loss: 2918.9241 - mae:
Epoch 6/10
38/38 [=====] - 40s 1s/step - loss: 2863.3491 - mae:
Epoch 7/10
38/38 [=====] - 39s 1s/step - loss: 2814.8513 - mae:
Epoch 8/10
```

```

38/38 [=====] - 39s 1s/step - loss: 2782.5598 - mae:
Epoch 9/10
38/38 [=====] - 45s 1s/step - loss: 2733.4893 - mae:
Epoch 10/10
38/38 [=====] - 39s 1s/step - loss: 2772.5398 - mae:

```

```

# Evaluación del modelo ajustado en el conjunto de test reescalado
model_VGG16_test_lossv1, model_VGG16_test_mae_v1 = model_VGG16_adjusted.evaluate(X
print(f"Test Loss: {model_VGG16_test_lossv1}, Test MAE: {model_VGG16_test_mae_v1}"

```

```

12/12 [=====] - 11s 923ms/step - loss: 4668.1406 - m
Test Loss: 4668.140625, Test MAE: 34.522342681884766

```



Desarrollaria mas módelo con 224 x 224 pero es casi a 15/20 min por epoch y me quedo sin RAM

```

plt.style.use("ggplot")
plt.figure(figsize=(8, 6))
plt.plot(np.arange(0, 10), history_adjusted.history["loss"], label="train_loss")
plt.plot(np.arange(0, 10), history_adjusted.history["val_loss"], label="val_loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="upper right")

plt.show()

```



1. El entrenamiento del modelo sobre datos de imágenes reescaladas a 48x48 muestra una disminución progresiva tanto en la pérdida (`loss`) como en el error absoluto medio (`mae`) a lo largo de 10 épocas.
2. La pérdida en el conjunto de validación disminuyó de 6379.4058 a 5412.9795, y el MAE de validación también mostró una mejora, indicando que el modelo está aprendiendo efectivamente de los datos.
3. Aunque hubo una reducción consistente en los valores de pérdida y MAE, la variabilidad en el desempeño de validación (leve aumento en algunas épocas) sugiere espacio para optimizar hiperparámetros y posiblemente la arquitectura.
4. El modelo muestra capacidad de generalización al mejorar en el conjunto de validación, pero podría beneficiarse de técnicas adicionales de regularización o ajuste para reducir aún más el error y aumentar la precisión.

Se inicializó un modelo secuencial con tres capas densas y activaciones ReLU, finalizando con una capa de salida lineal para regresión. El modelo, compilado con el optimizador Adam y evaluado mediante MSE y MAE, está diseñado para capturar relaciones complejas dentro de datos tabulares.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping

# Inicializamos el modelo
model_tab = Sequential()

# Definimos las capas densas
model_tab.add(Dense(128, activation='relu', input_shape=(X_tab_train.shape[1],)))
model_tab.add(Dense(64, activation='relu'))
model_tab.add(Dense(32, activation='relu'))
model_tab.add(Dense(1, activation='linear'))

# Compilamos el modelo
model_tab.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Entreno el modelo con Early stopping para evitar overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model_tab.fit(
    X_tab_train, y_tab_train,
    validation_split=0.2,
    epochs=100,
    callbacks=[early_stopping],
    batch_size=32
)
```

```
Epoch 1/100
30/30 [=====] - 2s 17ms/step - loss: 5683.8271 - mae: 10.0000
Epoch 2/100
30/30 [=====] - 0s 6ms/step - loss: 4244.8101 - mae: 8.0000
Epoch 3/100
30/30 [=====] - 0s 6ms/step - loss: 3289.8748 - mae: 6.0000
Epoch 4/100
30/30 [=====] - 0s 7ms/step - loss: 2786.4446 - mae: 5.0000
Epoch 5/100
30/30 [=====] - 0s 7ms/step - loss: 2538.6411 - mae: 4.0000
Epoch 6/100
30/30 [=====] - 0s 6ms/step - loss: 2420.8425 - mae: 3.0000
Epoch 7/100
30/30 [=====] - 0s 7ms/step - loss: 2349.4385 - mae: 2.0000
Epoch 8/100
30/30 [=====] - 0s 8ms/step - loss: 2278.4448 - mae: 1.0000
Epoch 9/100
30/30 [=====] - 0s 8ms/step - loss: 2225.0117 - mae: 0.5000
Epoch 10/100
```



```
30/30 [=====] - 0s 9ms/step - loss: 2126.6416 - mae: 10.0000
Epoch 11/100
30/30 [=====] - 0s 7ms/step - loss: 2097.8899 - mae: 9.9999
Epoch 12/100
30/30 [=====] - 0s 6ms/step - loss: 2060.5315 - mae: 9.9999
Epoch 13/100
30/30 [=====] - 0s 4ms/step - loss: 2026.8022 - mae: 9.9999
Epoch 14/100
30/30 [=====] - 0s 3ms/step - loss: 1995.9612 - mae: 9.9999
Epoch 15/100
30/30 [=====] - 0s 4ms/step - loss: 2008.1035 - mae: 9.9999
Epoch 16/100
30/30 [=====] - 0s 4ms/step - loss: 1945.4622 - mae: 9.9999
Epoch 17/100
30/30 [=====] - 0s 4ms/step - loss: 1919.1449 - mae: 9.9999
Epoch 18/100
30/30 [=====] - 0s 4ms/step - loss: 1898.5424 - mae: 9.9999
Epoch 19/100
30/30 [=====] - 0s 3ms/step - loss: 1890.0677 - mae: 9.9999
Epoch 20/100
30/30 [=====] - 0s 4ms/step - loss: 1848.6709 - mae: 9.9999
Epoch 21/100
30/30 [=====] - 0s 4ms/step - loss: 1833.4834 - mae: 9.9999
Epoch 22/100
30/30 [=====] - 0s 4ms/step - loss: 1868.1221 - mae: 9.9999
Epoch 23/100
30/30 [=====] - 0s 3ms/step - loss: 1825.8241 - mae: 9.9999
Epoch 24/100
30/30 [=====] - 0s 3ms/step - loss: 1811.0815 - mae: 9.9999
Epoch 25/100
30/30 [=====] - 0s 4ms/step - loss: 1782.0923 - mae: 9.9999
Epoch 26/100
30/30 [=====] - 0s 4ms/step - loss: 1802.1482 - mae: 9.9999
Epoch 27/100
30/30 [=====] - 0s 4ms/step - loss: 1743.0642 - mae: 9.9999
Epoch 28/100
```