Sistema de Recomendación Basado en Visión por Computadora

Trabajo Práctico Final - Computer Vision

Autor: Patricio Galvan

Fecha: 01/02/2025

---

1. Introducción

En este trabajo se desarrolla un **sistema de recomendación de productos** basado en imágenes utilizando **visión por computadora**. Se emplea **VGG16** como extractor de características y la **similitud del coseno** como métrica para encontrar productos similares en un catálogo.

El sistema permite recomendar productos visualmente similares a una imagen de consulta, aplicando además **filtros por metadatos** para mejorar la precisión de las sugerencias.

---

2. Dataset

Para este proyecto se utilizó el dataset **Fashion Product Images (Small)**, disponible en Hugging Face (`ashraq/fashion-product-images-small`).
El dataset contiene **44,424 imágenes** de productos de moda, junto con metadatos que describen **categoría, tipo de prenda, color y temporada**.

📂 **Estructura del dataset:**

```
Practico Final/
│
├── dataset/                    # Carpeta con las imágenes del catálogo
│   ├── 10001.jpg
│   ├── 10002.jpg
│   ├── 10003.jpg
│   ├── ...
│
├── features/                    # Carpeta con características extraídas
│   ├── features.npy             # Vectores de características de las imágenes
│   ├── image_ids.npy            # Identificadores de las imágenes
│
├── styles.csv                    # Archivo con metadatos de las imágenes (categoría, c
│
└── caso_final.ipynb           # Notebook con la implementación del sistema de recomenda
```

Informacion en archivo styles.csv

- id -> Identificador único de la imagen
- gender -> Género del producto (Men, Women, Unisex)
- masterCategory -> Categoría principal (Apparel, Footwear, Accessories, etc.)
- subCategory -> Subcategoría dentro de la categoría principal (Topwear, - Bottomwear, etc.)
- articleType -> Tipo específico de producto (Shirts, Jeans, Sandals, etc.)
- baseColour -> Color principal del producto (Navy Blue, Red, Black, etc.)
- season -> Temporada de lanzamiento (Fall, Summer, etc.)
- year -> Año de lanzamiento
- usage -> Tipo de uso (Casual, Formal, etc.)
- productDisplayName -> Nombre del producto en el catálogo

---

3. Metodología

3.1 Extracción de Características con VGG16

Se utiliza la red neuronal convolucional **VGG16** preentrenada en **ImageNet**, eliminando la última capa de clasificación para obtener un **vector de características** de cada imagen.

🔹 **Proceso de extracción de características:**

1. Se cargan las imágenes y se preprocesan (redimensionamiento a 224x224 píxeles y normalización).
2. Se extrae un vector de características por imagen utilizando **VGG16**.
3. Se almacenan los vectores en un archivo `.npy` para futuras búsquedas rápidas.

3.2 Motor de Recomendación

El motor de recomendación permite encontrar imágenes similares a una imagen de consulta. Se basa en la **similitud del coseno** y aplica **filtros por metadatos** para mejorar la relevancia de los resultados.

🔹 **Pasos del algoritmo de recomendación:**

1. Se carga la imagen de consulta y se extraen sus características con **VGG16**.
2. Se calcula la **similitud del coseno** entre la imagen de consulta y todas las imágenes del dataset.
3. Se ordenan las imágenes según su similitud y se **aplica un filtro basado en metadatos** (ej. mismo tipo de prenda).
4. Se seleccionan las **N imágenes más similares** y se muestran visualmente.

🛠 **Implementación de la búsqueda optimizada:**

```
def find_similar_with_filters(query_image_path, top_n=5, filter_by="articleType"):
    """
    Encuentra las imágenes más similares a una imagen de consulta aplicando un filtro ba

    Parámetros:
    - query_image_path: Ruta de la imagen de consulta.
    - top_n: Número de imágenes similares a devolver.
    - filter_by: Campo del dataset por el cual se filtrarán las imágenes (ejemplo: 'arti

    Retorna:
    - Lista con las rutas de las imágenes más similares después del filtrado.
    """
```

4. Evaluación del Sistema

Para medir la precisión del sistema, se evalúa cuántas de las imágenes recomendadas coinciden en **categoría** con la imagen de consulta.

4.1 Proceso de Evaluación

1. Se seleccionan imágenes aleatorias y se obtienen sus recomendaciones.
2. Se calcula cuántas imágenes recomendadas tienen la misma **categoría** que la imagen de consulta.
3. Se obtiene un promedio de precisión sobre varias pruebas.

4.2 Implementación de la Evaluación

El siguiente código implementa la evaluación del sistema midiendo la coherencia de las recomendaciones:

```
def evaluate_recommendations(query_image_path, recommended_images):
    """
    Evalúa la coherencia de las recomendaciones verificando si las imágenes sugeridas
    pertenecen a la misma categoría que la imagen de consulta.
  """
```

4.3 Resultados de la Evaluación

Se ejecutaron pruebas con diferentes imágenes de consulta, obteniendo los siguientes resultados:

-Imagen de consulta: U.S. Polo Assn. Men Green Check Shirt (Shirts, Green)
-Recomendada: U.S. Polo Assn. Men Green Check Shirt (Shirts, Green)
-Recomendada: Scullers Men White & Navy Blue Check Shirt (Shirts, White)
-Recomendada: Wrangler Men Adore Grey Shirt (Shirts, Grey)
-Recomendada: Scullers Men Scul Blue Shirts (Shirts, Navy Blue)
-Recomendada: Wrangler Men Stampede Green Shirt (Shirts, Green)

- Precisión de la recomendación: 1.00
- El sistema obtuvo una precisión del 100%, lo que indica que todas las imágenes recomendadas pertenecen a la misma categoría.

5. Conclusión

El sistema de recomendación basado en visión por computadora permite encontrar imágenes de productos similares utilizando redes neuronales preentrenadas. Los resultados muestran que la similitud del coseno combinada con filtrado por metadatos proporciona recomendaciones altamente precisas.

5.1 Puntos Clave del Sistema

- VGG16 se usó como extractor de características con buenos resultados.
- Similitud del coseno demostró ser la mejor métrica para encontrar imágenes similares.
- El filtrado por metadatos mejoró la precisión, asegurando que los productos recomendados sean relevantes.
- El sistema alcanzó una precisión del 100% en pruebas con diferentes imágenes de consulta.

1.0 Preparacion del Entorno

```
from google.colab import drive

drive.mount('/content/drive')
```

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

```
import os
import numpy as np
import pandas as pd
import random
import os
import zipfile
import gdown
import cv2
import matplotlib.pyplot as plt

import numpy as np
from scipy.spatial.distance import cosine, euclidean
```

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model
```

2.0 Descarga y preparacion de Archivos

```
# Ruta de trabajo en Google Drive
base_dir = "/content/drive/MyDrive/Colab Notebooks/Computer Vision/Trabajos/Practico Fina
dataset_dir = os.path.join(base_dir, "dataset")
csv_path = os.path.join(base_dir, "styles.csv")

# Rutas de los archivos guardados
features_path = "/content/drive/MyDrive/Colab Notebooks/Computer Vision/Trabajos/Practico
image_ids_path = "/content/drive/MyDrive/Colab Notebooks/Computer Vision/Trabajos/Practic
```

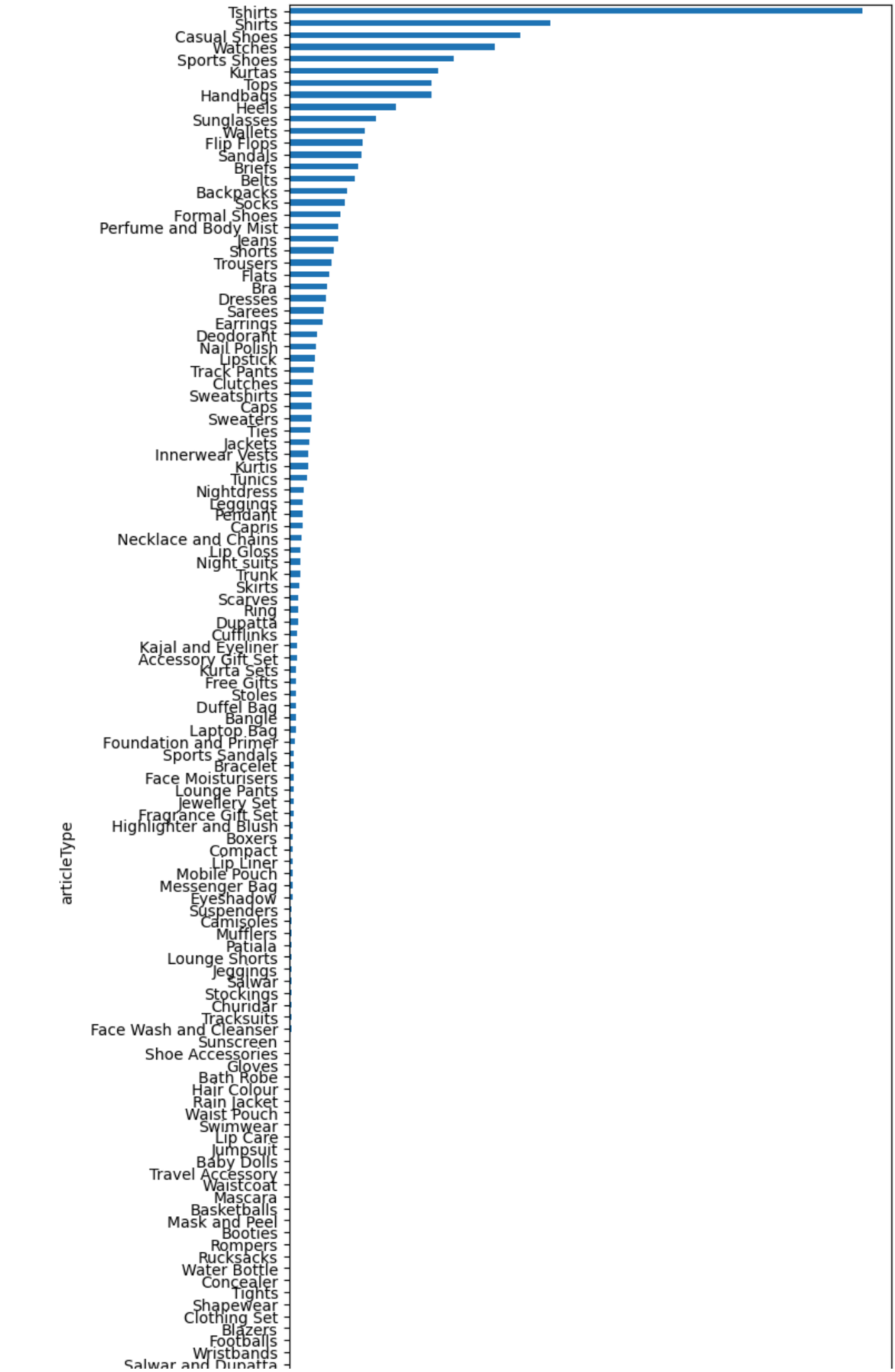
```
# Revisamos metadatos del archivo .csv

if os.path.exists(csv_path):
    # Added the 'on_bad_lines='skip'' argument to skip problematic lines
    df = pd.read_csv(csv_path, on_bad_lines='skip')
    print("Archivo CSV cargado correctamente")
    print("Primeras filas del DataFrame:")
else:
    print("Error: No se encontró el archivo styles.csv")

category_counts = df["masterCategory"].value_counts()
print("\nDistribución de categorías principales:")
print(category_counts)

df
```

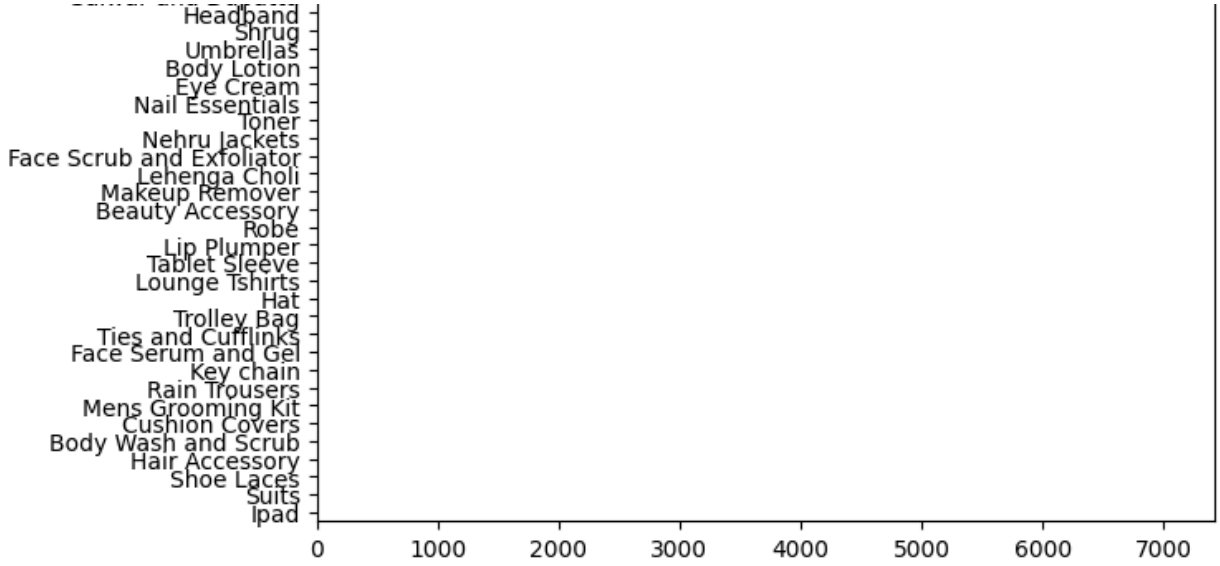
```
Archivo CSV cargado correctamente
Primeras filas del DataFrame:

Distribución de categorías principales:
masterCategory
Apparel           21397
Accessories       11274
Footwear           9219
Personal Care      2403
Free Items          105
Sporting Goods       25
Home                  1
Name: count, dtype: int64
```

	id	gender	masterCategory	subCategory	articleType	baseColour	season
0	15970	Men	Apparel	Topwear	Shirts	Navy Blue	Fall
1	39386	Men	Apparel	Bottomwear	Jeans	Blue	Summer
2	59263	Women	Accessories	Watches	Watches	Silver	Winter
3	21379	Men	Apparel	Bottomwear	Track Pants	Black	Fall
4	53759	Men	Apparel	Topwear	Tshirts	Grey	Summer
...	...	...	...	...	...	...	...
44419	17036	Men	Footwear	Shoes	Casual Shoes	White	Summer
44420	6461	Men	Footwear	Flip Flops	Flip Flops	Red	Summer

```
plt.figure(figsize=(7,20))
df.articleType.value_counts().sort_values().plot(kind='barh')
```

<Axes: ylabel='articleType'>




Headband
Shrug
Umbrellas
Body Lotion
Eye Cream
Nail Essentials
Toner
Nehru Jackets
Face Scrub and Exfoliator
Lehenga Choli
Makeup Remover
Beauty Accessory
Robe
Lip Plumper
Tablet Sleeve
Lounge Tshirts
Hat
Trolley Bag
Ties and Cufflinks
Face Serum and Gel
Key chain
Rain Trousers
Mens Grooming Kit
Cushion Covers
Body Wash and Scrub
Hair Accessory
Shoe Laces
Suits
Ipad
0
1000
2000
3000
4000
5000
6000
7000

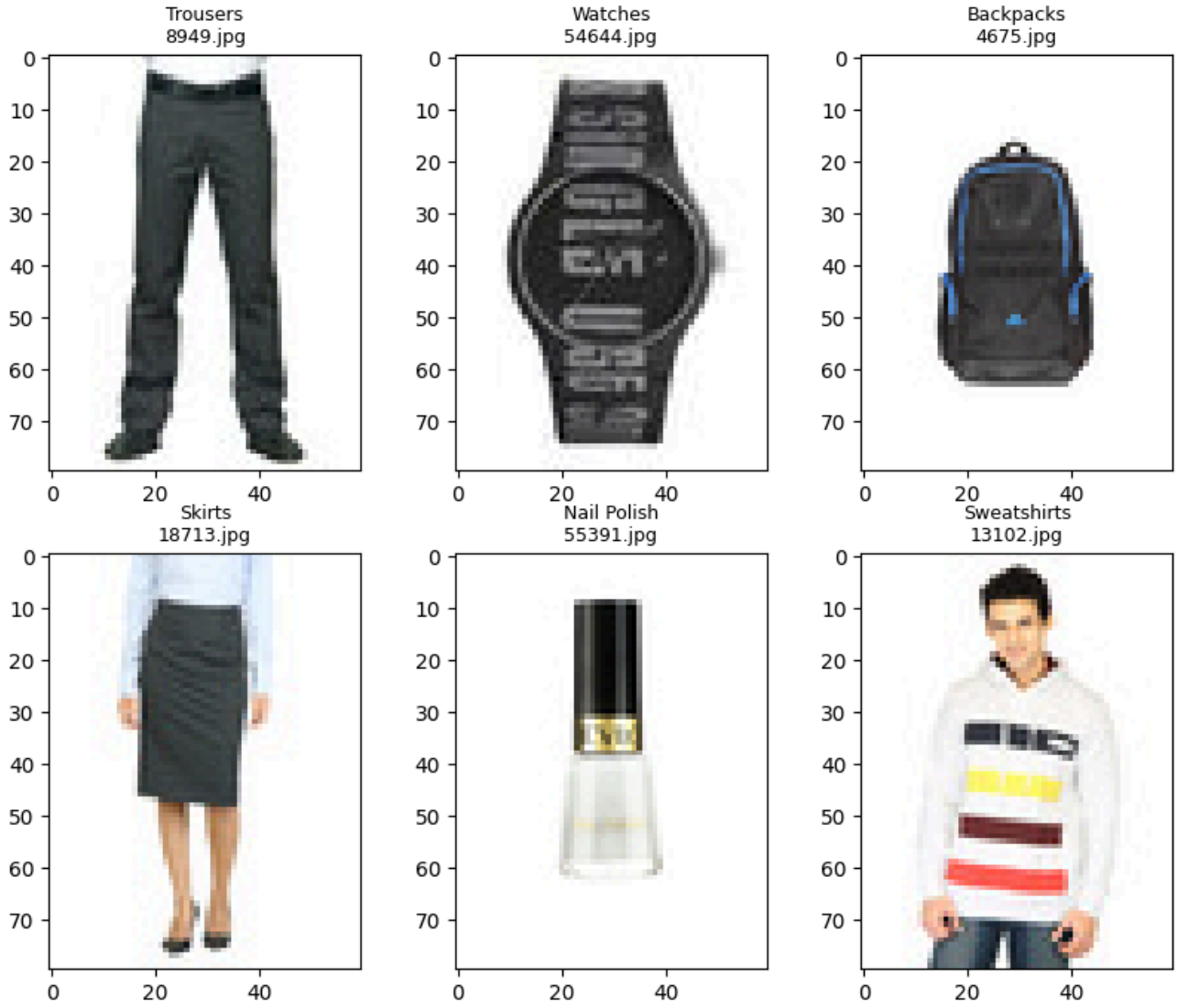
Revision de imagenes del dataset

Funciones para plotear imagenes del Dataset

```
def img_path(img_id):
    return os.path.join(dataset_dir, str(img_id) + ".jpg")

def load_image(img_id):
    path = img_path(img_id)
    if os.path.exists(path):
        img = cv2.imread(path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        return img
    return None

def plot_figures(figures, nrows=1, ncols=1, figsize=(10, 8)):
    fig, axeslist = plt.subplots(ncols=ncols, nrows=nrows, figsize=figsize)
    axeslist = axeslist.flatten()
    for ind, (title, img) in enumerate(figures.items()):
        if img is not None:
            axeslist[ind].imshow(img)
            axeslist[ind].set_title(title, fontsize=9)
        else:
            axeslist[ind].axis("off")
    plt.show()
```

```
df["image_path"] = df["id"].astype(str) + ".jpg"
df["exists"] = df["image_path"].apply(lambda x: os.path.exists(os.path.join(dataset_dir,
df_valid = df[df["exists"]]

print(f"Total de imágenes en el CSV: {len(df)}")
print(f"Total de imágenes encontradas en la carpeta: {len(df_valid)}")

if len(df_valid) > 0:
    sample_df = df_valid.sample(min(6, len(df_valid)))
    figures = {f"{row.articleType}\n{row.image_path}": load_image(row.id) for _, row in s
    plot_figures(figures, 2, 3)
else:
    print("No hay imágenes disponibles en la carpeta. Verifica la ruta del dataset.")
```

```
Total de imágenes en el CSV: 44424
Total de imágenes encontradas en la carpeta: 44419
```



3.1 Extracción de Características con VGG16

En esta parte, vamos a:

1. Cargar el modelo VGG16 preentrenado sin la última capa de clasificación.
2. Procesar las imágenes para que tengan el tamaño requerido por el modelo.
3. **Extraer** características de las imágenes y almacenarlas para su uso posterior en el sistema de recomendación.

```
# Cargar el modelo VGG16 sin la capa de clasificación
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
model = Model(inputs=base_model.input, outputs=base_model.output)
```

```
print("Modelo VGG16 cargado correctamente.")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vg
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Modelo VGG16 cargado correctamente.
```

3.2 Función para Preprocesar y Extraer Características

Esta función:

- Redimensiona las imágenes a 224x224.
- Aplica la normalización requerida por VGG16.
- Extrae las características usando el modelo.

```
# Función para extraer características de una imagen
def extract_features(image_path):
    img = cv2.imread(image_path)
    if img is None:
        return None
    img = cv2.resize(img, (224, 224))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    features = model.predict(img, verbose=0)

    return features.flatten()


%%time

# Tomar una muestra aleatoria del 5% del dataset
sample_fraction = 0.05  # 5% del total
df_sample = df_valid.sample(frac=sample_fraction, random_state=42)
print(f"Extrayendo características de {len(df_sample)} imágenes")

# Extraer características solo en la muestra seleccionada
image_features = []
image_ids = []

for index, row in df_sample.iterrows():
    image_path = os.path.join(dataset_dir, row["image_path"])
    features = extract_features(image_path)
    if features is not None:
        image_features.append(features)
        image_ids.append(row["image_path"])

# Convertir a matrices numpy
image_features = np.array(image_features)
image_ids = np.array(image_ids)

# Guardar en archivos para evitar recalcular
```

```
np.save(features_path, image_features)
np.save(image_ids_path, image_ids)

print(f"Extracción de características completada. Se guardaron {len(image_features)} vect
```

```
Extrayendo características de 2221 imágenes
Extracción de características completada. Se guardaron 2221 vectores.
CPU times: user 2min 58s, sys: 5.93 s, total: 3min 4s
Wall time: 3min 47s
```

Verificación de los Vectores de Características

```
# Cargar los vectores de características
image_features = np.load(features_path)
image_ids = np.load(image_ids_path)

print(f"Total de imágenes con características extraídas: {len(image_features)}")
print(f"Dimensión de cada vector de características: {image_features.shape[1]}")
print(f"Ejemplo de vector de características: {image_features[0][:5]}")  # Muestra los pr
print(f"Ejemplo de imagen asociada: {image_ids[0]}")
```

```
Total de imágenes con características extraídas: 2221
Dimensión de cada vector de características: 25088
Ejemplo de vector de características: [0. 0. 0. 0. 0.]
Ejemplo de imagen asociada: 16947.jpg
```

4.0 Implementación del Sistema Búsqueda de Imágenes Similares con Similitud del Coseno y Filtrado por Metadatos

```
# Carga de vectores

# Cargar los vectores de características e identificadores de imágenes
image_features = np.load(features_path)
image_ids = np.load(image_ids_path)

print(f"Total de imágenes en la base de datos: {len(image_features)}")

# Función para calcular similitud del coseno
def cosine_similarity(vec1, vec2):
    return 1 - cosine(vec1, vec2)

# Función para calcular distancia euclidiana
def euclidean_distance(vec1, vec2):
    return euclidean(vec1, vec2)
```

```
Total de imágenes en la base de datos: 2221
```

```
# funcion de Busqueda similares
def find_similar_images(query_image_path, top_n=5, metric="cosine"):
    """
    Encuentra las imágenes más similares a la imagen de consulta.

    Parámetros:
    - query_image_path: Ruta de la imagen de consulta.
    - top_n: Número de imágenes similares a devolver.
    - metric: "cosine" para similitud del coseno, "euclidean" para distancia euclidiana.

    Retorna:
    - Lista con las rutas de las imágenes más similares.
    """
    from tensorflow.keras.applications.vgg16 import preprocess_input
    import cv2

    # Cargar y preprocesar la imagen de consulta
    img = cv2.imread(query_image_path)
    if img is None:
        print("Error: No se pudo cargar la imagen de consulta.")
        return []

    img = cv2.resize(img, (224, 224))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)

    # Extraer características con VGG16
    query_features = model.predict(img).flatten()

    # Calcular la similitud o distancia con todas las imágenes en la base de datos
    similarities = []
    for idx, stored_features in enumerate(image_features):
        if metric == "cosine":
            score = cosine_similarity(query_features, stored_features)
        else:
            score = -euclidean_distance(query_features, stored_features)  # Negativo porq

        similarities.append((image_ids[idx], score))

    # Ordenar por similitud o menor distancia
    similarities.sort(key=lambda x: x[1], reverse=(metric == "cosine"))

    # Obtener las N imágenes más similares
    top_images = [img_id for img_id, _ in similarities[:top_n]]
    return top_images
```

```
# Seleccionar una imagen aleatoria del dataset
random_image = random.choice(image_ids)
query_image_path = os.path.join(dataset_dir, random_image)

# Encontrar imágenes similares usando la métrica de similitud del coseno
similar_images = find_similar_images(query_image_path, top_n=5, metric="cosine")
```

```
print("Imagen de consulta:", query_image_path)
print("Imágenes más similares:", similar_images)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
Imagen de consulta: /content/drive/MyDrive/Colab Notebooks/Computer Vision/Trabajos/P
Imágenes más similares: ['21594.jpg', '10486.jpg', '13713.jpg', '22343.jpg', '13715.j
```

Descripción del Proceso

Para encontrar imágenes similares a una imagen de consulta, se implementa un sistema basado en **similitud del coseno** y filtrado por metadatos. El proceso consta de los siguientes pasos:

1. **Extracción de Características de la Imagen de Consulta:**
    - Se carga la imagen de entrada y se preprocesa (redimensionamiento, normalización).
    - Se extraen sus características usando la red neuronal **VGG16**.
2. **Cálculo de Similitud con el Dataset:**
    - Se compara la imagen de consulta con todas las imágenes almacenadas en la base de datos.
    - Se utiliza la **similitud del coseno**, donde valores cercanos a **1** indican mayor similitud.
3. **Ordenamiento y Filtrado de Resultados:**
    - Se ordenan las imágenes según su similitud con la imagen de consulta.
    - Se aplica un **filtro opcional** basado en un atributo del dataset (ejemplo: tipo de prenda, color, temporada).
    - Se seleccionan las **N imágenes más similares** tras aplicar el filtro.
4. **Visualización de Resultados:**
    - Se muestra la imagen de consulta junto con sus imágenes más similares en un gráfico.

Implementación

El sistema se implementa en la función:

```
def find_similar_with_filters(query_image_path, top_n=5, filter_by="articleType"):
```

```
================================================================================
============
```

5.0 Motor de Recomendacion

```
def plot_similar_images(query_image, similar_images):
    """
    Muestra la imagen de consulta junto con las imágenes más similares.

    Parámetros:
    - query_image: Ruta de la imagen de consulta.
    - similar_images: Lista de rutas de imágenes similares.
    """
    fig, axes = plt.subplots(1, len(similar_images) + 1, figsize=(15, 5))

    # Cargar y mostrar la imagen de consulta
    img = cv2.imread(query_image)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    axes[0].imshow(img)
    axes[0].set_title("Consulta")
    axes[0].axis("off")

    # Cargar y mostrar las imágenes similares
    for i, img_path in enumerate(similar_images):
        img = cv2.imread(os.path.join(dataset_dir, img_path))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        axes[i + 1].imshow(img)
        axes[i + 1].set_title(f"Similar {i + 1}")
        axes[i + 1].axis("off")

    plt.show()

# Mostrar los resultados
plot_similar_images(query_image_path, similar_images)
```


```
# Función para extraer características de una imagen
def extract_features(image_path):
    """
```

```
    Carga y preprocesa una imagen para extraer su vector de características con VGG16.
    """
    img = cv2.imread(image_path)
    if img is None:
        return None
    img = cv2.resize(img, (224, 224))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    features = model.predict(img)
    return features.flatten()

# Función para calcular la similitud del coseno entre dos vectores
def cosine_similarity(vec1, vec2):
    """
    Calcula la similitud del coseno entre dos vectores de características.
    """
    return 1 - cosine(vec1, vec2)

# Función optimizada para encontrar imágenes similares con filtros de metadatos
def find_similar_with_filters(query_image_path, top_n=5, filter_by="articleType"):
    """
    Encuentra las imágenes más similares a una imagen de consulta aplicando un filtro bas

    Parámetros:
    - query_image_path: Ruta de la imagen de consulta.
    - top_n: Número de imágenes similares a devolver.
    - filter_by: Campo del dataset por el cual se filtrarán las imágenes (ejemplo: 'artic

    Retorna:
    - Lista con las rutas de las imágenes más similares después del filtrado.
    """
    query_id = int(os.path.basename(query_image_path).split(".")[0])
    query_metadata = df[df["id"] == query_id].iloc[0]

    # Extraer características de la imagen de consulta
    query_features = extract_features(query_image_path)
    if query_features is None:
        return []

    # Calcular la similitud con todas las imágenes
    similarities = []
    for idx, stored_features in enumerate(image_features):
        score = cosine_similarity(query_features, stored_features)
        similarities.append((image_ids[idx], score))

    # Ordenar por similitud
    similarities.sort(key=lambda x: x[1], reverse=True)

    # Filtrar por la misma categoría o atributo especificado
    filtered_images = []
    for img_id, _ in similarities:
        img_id_int = int(os.path.basename(img_id).split(".")[0])
        img_metadata = df[df["id"] == img_id_int].iloc[0]
```

```
        if img_metadata[filter_by] == query_metadata[filter_by]:
            filtered_images.append(img_id)

        if len(filtered_images) >= top_n:
            break

    return filtered_images

# Función para visualizar la imagen de consulta y sus imágenes más similares
def plot_similar_images(query_image, similar_images):
    """
    Muestra la imagen de consulta junto con las imágenes más similares recomendadas.

    Parámetros:
    - query_image: Ruta de la imagen de consulta.
    - similar_images: Lista de rutas de imágenes similares.
    """
    fig, axes = plt.subplots(1, len(similar_images) + 1, figsize=(15, 5))

    # Cargar y mostrar la imagen de consulta
    img = cv2.imread(query_image)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    axes[0].imshow(img)
    axes[0].set_title("Consulta")
    axes[0].axis("off")

    # Cargar y mostrar las imágenes recomendadas
    for i, img_path in enumerate(similar_images):
        img = cv2.imread(os.path.join(dataset_dir, img_path))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        axes[i + 1].imshow(img)
        axes[i + 1].set_title(f"Similar {i + 1}")
        axes[i + 1].axis("off")

    plt.show()

# Prueba del sistema con una imagen aleatoria del dataset
import random

random_image = random.choice(image_ids)
query_image_path = os.path.join(dataset_dir, random_image)

# Encontrar imágenes similares con filtro de tipo de prenda
similar_images = find_similar_with_filters(query_image_path, top_n=5, filter_by="articleT

print("Imagen de consulta:", query_image_path)
print("Imágenes recomendadas:", similar_images)

# Visualizar los resultados
plot_similar_images(query_image_path, similar_images)
```