Sistema de Recuperación de Imágenes Basado en Contenido (CBIR)

Daniel Llopis, Alba Sanz
Algoritmos y Arquitecturas para Procesado de Imágenes
Email: d.llopis@alumnos.upm.es, alba.sanz.bustos@alumnos.upm.es

***Abstract*—En el mundo actual, la cantidad de imágenes digitales está creciendo exponencialmente. Esto ha creado la necesidad de sistemas eficientes para buscar y recuperar imágenes basadas en su contenido visual en lugar de solo metadatos o etiquetas textuales.**

Este informe presenta el desarrollo de un sistema CBIR sobre un dataset de comida rápida con 6 clases que utiliza diferentes métodos de extracción de características y técnicas de indexación para permitir una búsqueda eficiente y precisa de imágenes.

***Index Terms*—CBIR, extracción de características, FAISS, Streamlit, visión por computadora, métricas de evaluación**

I. Introducción

La cantidad de imágenes digitales sigue creciendo debido al uso masivo de dispositivos móviles y plataformas en línea, lo que demanda sistemas de búsqueda que utilicen el contenido visual como criterio. Los sistemas de Recuperación de Imágenes Basada en Contenido (CBIR, por sus siglas en inglés) abordan este desafío al permitir que los usuarios realicen búsquedas utilizando una imagen como consulta y recuperando imágenes visualmente similares.

Este informe describe el desarrollo de un sistema CBIR con métodos de extracción de características, indexación, interfaz de usuario y evaluación.

II. Descripción del Sistema CBIR

A. Visión General

El sistema CBIR desarrollado consta de varios componentes clave:

- **Dataset de imágenes**: Organizado en categorías para la evaluación de relevancia.
- **Extracción de características**: Cinco métodos diferentes para representar imágenes numéricamente.
- **Indexación**: Uso de FAISS para crear índices que permitan búsquedas rápidas y eficientes.
- **Interfaz de usuario**: Una aplicación basada en Streamlit que permite a los usuarios cargar imágenes de consulta y obtener resultados.
- **Evaluación del rendimiento**: Métricas como Precisión, Recall, Average Precision (AP) y F1 Score para medir la eficacia del sistema..

III. Dataset y Preprocesamiento

El dataset utilizado está organizado en subcarpetas, donde cada subcarpeta representa una categoría o clase. Esta estructura permite extraer fácilmente las etiquetas de cada imagen, lo que facilita la evaluación de la relevancia de los resultados recuperados.

Para estandarizar el proceso de evaluación, se generó un archivo `labels.csv` que asocia cada imagen con su etiqueta correspondiente. Este archivo incluye dos columnas principales: el nombre de la imagen y su clase. Es fundamental para calcular las métricas de evaluación, como precisión y recall, y para realizar análisis del rendimiento del sistema. A continuación, se muestra un fragmento del archivo `labels.csv`:

TABLE I
EJEMPLO DEL ARCHIVO `labels.csv`

Imagen	Etiqueta
Donut/Donut (44).jpeg	Donut
Donut/Donut (7).jpeg	Donut
Donut/Donut (1254).jpeg	Donut
Burguer/005.jpg	Burguer
Burguer/007.jpg	Burguer
Burguer/303.jpg	Burguer

El archivo `labels.csv` fue generado automáticamente mediante un script Python, que recorrió las subcarpetas del dataset y extrajo las etiquetas a partir de los nombres de las carpetas. Este proceso aseguró consistencia en la organización y simplificó la evaluación posterior. ESTO NO SE SI ES VERDAD O NO

IV. Métodos de Extracción de Características

A. Extractor 1: Histograma de Color

Este método usa **histogramas en el espacio de color HSV** para capturar la distribución de colores.

- Cargar y redimensionar la imagen a un tamaño uniforme (224x224 píxeles).
- Convertir la imagen al espacio de color HSV.
- Calcular el histograma de color en los canales H, S y V con 8 bins cada uno.
- Normalizar el histograma para que la suma de sus valores sea 1.

B. Extractor 2: CNN Pre-entrenada (VGG16)

Este extractor utiliza una **red neuronal convolucional pre-entrenada (VGG16)** para extraer características de alto nivel de las imágenes.

- Cargar el modelo VGG16 sin las capas superiores y con un pooling global máximo.
- Preprocesar la imagen para que sea compatible con el modelo (redimensionar y normalizar).
- Extraer características pasando la imagen por el modelo.

C. Extractor 3: Histogramas de Textura (LBP)

Este extractor utiliza el patrón binario local **(LBP)** para capturar la textura de las imágenes.

- Cargar la imagen en escala de grises y redimensionarla.
- Calcular el patrón LBP utilizando parámetros definidos (radio y número de puntos).
- Calcular el histograma del patrón LBP.
- Normalizar el histograma.

D. Extractor 4: HOG (Histogram of Oriented Gradients)

El **método HOG** es ampliamente utilizado para capturar características relacionadas con los contornos y la forma de los objetos en una imagen.

- Convertir la imagen a escala de grises.
- Dividir la imagen en celdas pequeñas (e.g., 8x8 píxeles).
- Calcular el gradiente horizontal y vertical en cada celda.
- Generar un histograma de gradientes orientados para cada celda.
- Normalizar los histogramas en bloques de celdas.
- Combinar los histogramas en un único vector de características.

E. Extractor 5: ORB (Oriented FAST and Rotated BRIEF)

ORB combina:

- FAST para la detección de puntos clave.
- BRIEF como descriptor, adaptado para robustez frente a cambios de orientación.

V. Indexación con FAISS

FAISS (Facebook AI Similarity Search) es una biblioteca diseñada para realizar búsquedas eficientes en grandes conjuntos de datos vectoriales. En este proyecto, se utilizó FAISS para crear índices de los vectores de características extraídos, permitiendo búsquedas rápidas y precisas basadas en similitud.

A. Tipos de índices

FAISS ofrece múltiples métodos de indexación, cada uno con características y aplicaciones específicas. Los principales métodos considerados en este proyecto fueron los siguientes:

- **Index Flat (Brute-force)**: Este método realiza una búsqueda exacta comparando el vector de consulta con todos los vectores del índice. Es el más simple de implementar y garantiza resultados precisos. Sin embargo, su complejidad temporal es lineal ($O(n)$) con respecto al número de vectores, lo que puede volverse ineficiente para grandes datasets.
- **Index IVF (Inverted File Index)**: Divide el espacio vectorial en clústeres utilizando algoritmos como K-Means. Durante la búsqueda, solo se consideran los clústeres más cercanos al vector de consulta, lo que reduce significativamente el número de comparaciones necesarias. Este método es especialmente útil para conjuntos de datos medianos o grandes, ya que mejora la eficiencia a costa de introducir aproximaciones.
- **Index PQ (Product Quantization)**: Divide cada vector en subvectores y los cuantiza para reducir su tamaño y memoria requerida. Esto permite manejar grandes volúmenes de datos con un uso de memoria más eficiente. Es ideal para búsquedas aproximadas en datasets masivos (más de 100,000 vectores).

B. Justificación de la elección

Para este proyecto, seleccionamos el índice **Index Flat** debido a las características de nuestro dataset y los objetivos del sistema:

- Nuestro dataset consta de menos de 1000 imágenes, lo que permite realizar búsquedas exhaustivas (*brute-force*) sin comprometer significativamente el tiempo de respuesta.
- Aunque exploramos **Index IVF**, su complejidad añadida no era necesaria para un conjunto de datos tan reducido. Además, el proceso de entrenamiento para agrupar los vectores en clústeres (como el algoritmo K-Means) añade un paso adicional que no aporta beneficios significativos en este caso.
- Por otro lado, **Index PQ** está diseñado para optimizar búsquedas en grandes datasets con limitaciones de memoria. Dado que nuestro conjunto de datos no enfrenta restricciones de este tipo, el uso de este método habría sido excesivamente complejo y podría introducir errores de aproximación innecesarios.

La decisión de utilizar **Index Flat** se basó en su simplicidad y en que garantiza resultados exactos sin necesidad de aproximaciones. Si el tamaño del dataset aumentara significativamente en futuros desarrollos, sería pertinente reconsiderar el uso de Index IVF o Index PQ para equilibrar precisión, memoria y tiempo de búsqueda.

VI. Implementación del Sistema CBIR

La implementación del sistema CBIR se llevó a cabo mediante dos scripts principales: `create_index.py` para la creación de índices y `app.py` para la interfaz de usuario. Además, se establecieron requisitos técnicos específicos para garantizar el correcto funcionamiento del sistema. A continuación, se describe cada componente en detalle.

A. Creación y Entrenamiento de Índices

El script `create_index.py` se encargó de gestionar todo el proceso relacionado con los índices. En este script se realizan las siguientes tareas:

- Agregar los vectores de características al índice.
- Guardar el índice para uso futuro.

Adicionalmente, este script realiza las siguientes acciones clave:

- Extrae las características de todas las imágenes utilizando los extractores definidos.
- Genera y guarda los índices para cada extractor y tipo de índice.
- Crea los archivos auxiliares `db.csv` y `labels.csv`, que contienen la información necesaria para la evaluación y el mapeo de etiquetas.

B. Interfaz de Usuario con Streamlit

La interfaz de usuario se implementó en el script `app.py`, utilizando Streamlit para crear una aplicación interactiva. Esta aplicación permite al usuario:

- Seleccionar el extractor de características deseado.
- Subir una imagen de consulta e introducir su clase correspondiente.
- Visualizar los resultados de búsqueda, junto con las métricas de evaluación del sistema.

C. Requisitos de Implementación

Para garantizar el correcto funcionamiento del sistema, se establecieron los siguientes requisitos técnicos:

- Crear un entorno virtual para gestionar las dependencias del proyecto.
- Seguir la estructura de carpetas indicada, que debe incluir:

TABLE II
ESTRUCTURA DEL PROYECTO

Archivo/Carpeta	Descripción
`app.py`	Archivo principal: interfaz del sistema CBIR
`create_index.py`	Script para crear índices y extraer características
`images/`	Carpeta con imágenes organizadas en categorías
`README.md`	Detalles del proyecto y documentación general
`requirements.txt`	Dependencias necesarias para ejecutar el sistema

El archivo `requirements.txt` incluye todas las bibliotecas necesarias para ejecutar el sistema, como `FAISS`, `Streamlit`, y bibliotecas estándar como `numpy` y `pandas`.

VII. MÉTRICAS DE EVALUACIÓN

Las métricas utilizadas para medir el rendimiento del sistema incluyen:

- **Precisión@10**: Proporción de imágenes relevantes entre las recuperadas.
- **Recall@10**: Proporción de imágenes relevantes recuperadas respecto al total de relevantes.
- **Average Precision (AP)**: Promedio de precisiones en posiciones relevantes.
- **F1 Score**: Media armónica entre precisión y recall.

VIII. RESULTADOS Y ANÁLISIS

Decidimos medir la eficacia de los extractores principalmetne mediante el accuracy, dado que mide qué porcentaje de imágenes fueron clasificadas correctamente en sus respectivas clases.
En este proyecto CBIR, donde el objetivo principal es recuperar imágenes relevantes, un alto accuracy indica que el sistema puede identificar correctamente las categorías de las imágenes, lo que es crucial para aplicaciones como búsqueda de imágenes similares o clasificación automática de imágenes en categorías relevantes.
Además, es fácil de entender e intuitivo para comparar, convirtiéndose así en una métrica consistente para evaluar el rendimiento en un escenario multiclase, lo que puede ser más difícil con métricas como el F1-Score, que requiere análisis por clase.

Sin embargo, cabe tener en cuenta que en CBIR, el accuracy puede no ser suficiente en estos escenarios:
1. Clases desequilibradas: Si algunas clases tienen muchas más imágenes que otras, métricas como el F1-Score o el Mean Average Precision (mAP) podrían ser más informativas.
2. Tareas de ranking: En tareas donde el orden de las imágenes recuperadas importa más que la clasificación estricta, métricas como Precision@K o Recall@K son más relevantes.

Se realizaron pruebas con los cinco métodos de extracción, obteniendo los siguientes resultados:

TABLE III
PRUEBAS DE ACCURACY POR CLASE PARA CADA EXTRACTOR

Extractor	Burguer	Rice	Donut	C.Chicken	B.Potato	Pizza
HC	0.00	0.40	0.50	0.50	0.00	0.70
CNN	1.00	1.00	0.80	0.70	0.60	0.80
HT	0.20	0.20	0.50	0.00	0.30	0.00
HOG	0.00	0.80	0.00	0.10	0.00	0.40
ORB	0.20	0.00	0.50	0.30	0.30	0.00

El método basado en CNN obtuvo el mejor rendimiento, capturando características de alto nivel más representativas del contenido visual.

Clases como “Burger” y “Baked Potato” son particularmente difíciles de clasificar para casi todos los extractores, excepto para CNN en “Burger”. Por tanto: CNN es la mejor opción para este conjunto de datos, y sería interesante centrarse en optimizar este extractor. Los extractores tradicionales (HC, HOG, ORB, HT) podrían no ser adecuados para este caso específico debido a la falta de discriminación entre clases. Para mejorar el rendimiento general, podríamos considerar combinar los extractores (e.g., concatenar características) o ajustar los parámetros de los métodos que muestran un desempeño moderado.

URL de carpeta con ejemplos de la ejecución : https://drive.google.com/drive/folders/100SOJCKGEDWvpZfbbOtJLrHopEd_ksH?usp=drive_link

IX. Conclusiones

El sistema CBIR desarrollado demuestra la eficacia de utilizar diferentes métodos de extracción de características e indexación para la recuperación de imágenes basada en contenido. Los resultados muestran que las características extraídas con CNN pre-entrenadas ofrecen el mejor rendimiento, lo que es consistente con la literatura actual en visión por computadora. La estructura del dataset y el uso de FAISS permiten una búsqueda eficiente.

X. Hallazgos clave

- Uso de CNN Pre-entrenadas: Las CNN capturan características de alto nivel que mejoran significativamente el rendimiento en comparación con métodos basados en características de bajo nivel.
- Importancia de la Estructura del Dataset: Tener imágenes bien categorizadas y etiquetadas es crucial para evaluar el rendimiento y calcular las métricas de manera precisa.
- Eficiencia de FAISS: El uso de FAISS permite realizar búsquedas rápidas incluso con índices que requieren entrenamiento, como IVF y PQ.

XI. Trabajo Futuro

- Integración de Más Extractores: Implementar y probar otros métodos de extracción de características, como SIFT o SURF.
- Ajuste de Parámetros: Optimizar los parámetros de los extractores y los índices para mejorar aún más el rendimiento.
- Ampliación del Dataset: Utilizar datasets más grandes y diversos para evaluar la escalabilidad y robustez del sistema.
- Mejora de la Interfaz de Usuario: Añadir funcionalidades como filtros por categoría o visualización de las similitudes.

Appendix A
Código Fuente

El código fuente completo de este proyecto está disponible en el siguiente enlace de GitHub: https://github.com/danillopis/CBIR

References

[1] FAISS: Facebook AI Similarity Search. Disponible en: https://github.com/facebookresearch/faiss

[2] Streamlit: Framework para la creación de aplicaciones web interactivas en Python. Disponible en: https://streamlit.io/

[3] OpenCV: Biblioteca de visión por computadora. Disponible en: https://opencv.org/

[4] TensorFlow Keras Applications: Modelos pre-entrenados disponibles en Keras. Disponible en: https://keras.io/api/applications/