ESCUELA
POLITÉCNICA
NACIONAL

FACULTAD DE INGENIERÍA EN SISTEMAS

CARRERRA DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

RECUPERACIÓN DE LA INFORMACIÓN

PROYECTO BIMESTRAL

SISTEMA DE RECUPERACIÓN BASADO EN IMÁGENES

INTEGRANTES

PAOLA AUCAPIÑA

KAREN GUAÑA

PROF. IVÁN CARRERA

FECHA DE ENTREGA

08/08/2024

Contenido

1. INTRODUCCIÓN

El objetivo de este trabajo es diseñar y desarrollar una máquina de búsqueda que permita a los usuarios realizar consultas utilizando imágenes en lugar de texto. Este sistema debe ser capaz de encontrar imágenes similares dentro de una base de datos dada. El proyecto se dividirá en varias fases, que se describen a continuación.

2. FASES DEL PROYECTO

2.1 ADQUISICIÓN DE DATOS

Se inició con la descarga del dataset con nombre 'Caltech 101' para lo cual se ejecutó el siguiente código:

```
# Directorio para la descarga
data_dir = r'C:\Users\Paola\Downloads\caltech101'

# Descargar y cargar el dataset
(train_dataset, test_dataset), dataset_info = tfds.load(
    name='caltech101:3.0.2',
    split=['train[:80%]', 'test[:20%]'],
    with_info=True,
    as_supervised=True,
    data_dir=data_dir,
    download=False # Cambiar a True para descargar
)
```

Figura 1. Descarga

2.2 PREPROCESAMIENTO

En este paso, se lleva a cabo el preprocesamiento de las imágenes antes de ser introducidas en el modelo de clasificación. Este proceso es fundamental para asegurar que las imágenes tengan un formato y tamaño uniforme, lo cual es necesario para un entrenamiento eficiente y preciso del modelo.

Redimensionamiento de Imágenes:

Se redimensionan todas las imágenes a un tamaño de 224x224 píxeles utilizando la función tf.image.resize(). Este tamaño ha sido elegido para ser compatible con la arquitectura de la red neuronal que se utilizará

Normalización de las Imágenes:

Después del redimensionamiento, las imágenes son convertidas a un formato de punto flotante y sus valores de píxel se normalizan al rango [0, 1]. Esto se logra dividiendo los valores de los píxeles (que originalmente están en el rango [0, 255]) por 255.

```python
# Preprocesar imágenes
def preprocess_image(image, label):
    image = tf.image.resize(image, (224, 224))  # InceptionV3 usa 299x299
    image = tf.cast(image, tf.float32) / 255.0
    return image, label
```

Figura 2. Preprocesamiento

Preparación de los Conjuntos de Datos:
Conjunto de Entrenamiento: Las imágenes preprocesadas del conjunto de entrenamiento se mezclan aleatoriamente (shuffle(1000)) para asegurar que el modelo no se vea influenciado por el orden de los datos.
Conjunto de Prueba: Similar al conjunto de entrenamiento, las imágenes del conjunto de prueba se agrupan en lotes de 32 imágenes para su evaluación posterior.

```python
train_dataset = train_dataset.map(preprocess_image).shuffle(1000).batch(32)
test_dataset = test_dataset.map(preprocess_image).batch(32)
```

Figura 3. Conjunto datos

2.3 EXTRACCIÓN DE CARACTERÍSTICAS

En este apartado se configura un modelo preentrenado y se utiliza para extraer características de las imágenes, lo cual es un paso crucial en muchos sistemas de visión por computadora.
Configuración del Modelo Preentrenado
Se utiliza el modelo VGG16 preentrenado con la base de datos ImageNet. Este modelo es ampliamente reconocido por su capacidad para extraer características ricas y complejas de las imágenes.
Extracción de Características
Se define una función extract_features() para extraer características de las imágenes en los conjuntos de datos de entrenamiento y prueba. Esta función también guarda las imágenes y las etiquetas correspondientes.
Proceso de Extracción:

- Las imágenes se pasan a través del modelo preentrenado, generando mapas de características (feature_maps) para cada imagen.
- Estos mapas de características se almacenan junto con las etiquetas y las imágenes originales.
- Finalmente, la función devuelve tres listas: las características extraídas, las etiquetas y las imágenes originales.

```python
# Configurar el modelo InceptionV3
#base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299, 299, 3))
#model = Model(inputs=base_model.input, outputs=base_model.layers[-1].output)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
model = Model(inputs=base_model.input, outputs=base_model.layers[-1].output)

# Función para extraer características e imagenes
def extract_features(dataset):
    features = []
    labels = []
    imgs = []
    for images, lbls in dataset:
        imgs.append(images)
        feature_maps = model.predict(images)
        features.append(feature_maps)
        labels.append(lbls.numpy())
    return features, labels,imgs

# Extraer características para train y test
train_features, train_labels, train_img = extract_features(train_dataset)
test_features, test_labels, test_img = extract_features(test_dataset)
```

Figura 4. Modelo

2.4 INDEXACIÓN

Las características extraídas de las imágenes, que son inicialmente de alta dimensionalidad y están organizadas en lotes, se aplanan y se reorganizan en estructuras más manejables.

Aplanamiento de Características: Las características de cada imagen, que originalmente están en forma de matrices, se convierten en vectores (listas planas) utilizando el método flatten(). Esto simplifica la comparación y búsqueda entre imágenes.

Reorganización de Etiquetas e Imágenes: Las etiquetas y las imágenes también se reorganizan para que coincidan con las características aplanadas, asegurando que toda la información esté alineada correctamente.

Aplicación al Conjunto de Prueba: Se repite el mismo proceso para las imágenes del conjunto de prueba, manteniendo la consistencia en todo el dataset.

```python
#Indexación
# Aplanar las características para el índice
train_features_flat = np.array([feature.flatten() for batch in train_features for feature in batch])
train_labels_flat = np.array([label for batch in train_labels for label in batch])
train_img_flat = np.array([img for batch in train_img for img in batch])
#lineas nuevas para test
test_features_flat = np.array([feature.flatten() for batch in test_features for feature in batch])
test_labels_flat = np.array([label for batch in test_labels for label in batch])
test_img_flat = np.array([img for batch in test_img for img in batch])
```

Figura 5. Indexación

Una vez que hemos construido el índice, el siguiente paso es guardar los datos procesados en archivos para poder utilizarlos posteriormente sin necesidad de repetir el proceso de extracción y aplanamiento.

```
# Guardar el archivo
np.save('data/train_features.npy', train_features_flat)
np.save('data/train_labels.npy', train_labels_flat)
np.save('data/train_img.npy', train_img_flat)
#lineas nuevas para test
np.save('data/test_features.npy', test_features_flat)
np.save('data/test_labels.npy', test_labels_flat)
np.save('data/test_img.npy', test_img_flat)
```

Figura 6. Archivos guardados

2.5 DISEÑO DEL MOTOR DE BÚSQUEDA

Se carga un modelo preentrenado (VGG16 en este caso) para extraer características de las imágenes.
Se cargan las características, etiquetas, y rutas de las imágenes que ya fueron preprocesadas y guardadas en archivos .npy.
Se utiliza el modelo **NearestNeighbors** de sklearn para construir un índice basado en las características extraídas. Este modelo permite buscar las imágenes más similares a una imagen dada utilizando la métrica de distancia coseno.
Cuando un usuario sube una imagen, esta se preprocesa (redimensiona y normaliza) y se extraen sus características usando el modelo preentrenado.
El algoritmo de ranking se basa en las distancias calculadas por NearestNeighbors. Las imágenes se ordenan desde la más cercana (menor distancia) hasta la más lejana.
Las imágenes más similares encontradas se guardan temporalmente y se muestran en la interfaz de usuario, junto con sus etiquetas correspondientes.
El sistema está implementado como una aplicación web utilizando Flask.

```python
# Process the uploaded image
image = Image.open(image_file)
image = preprocess_image(image)
query_features = extract_features(image)


# Find the nearest neighbors
distances, indices = nn_model.kneighbors(query_features)
distances = distances[0]
indices = indices[0]
print("Indices of nearest neighbors:", indices)
print("Distances to nearest neighbors:", distances)

# Retrieve nearest images and labels using indices
nearest_images = [train_img_flat[i] for i in indices.flatten()]
nearest_labels = [train_labels_flat[i] for i in indices.flatten()]

# Save nearest images to a temporary folder
if not os.path.exists(app.config['RESULT_FOLDER']):
    os.makedirs(app.config['RESULT_FOLDER'])

for i, img in enumerate(nearest_images):
    img_pil = Image.fromarray((img * 255).astype(np.uint8))  # Convert back to uint8
    img_pil.save(os.path.join(app.config['RESULT_FOLDER'], f'neighbor_{i}.jpg'))
```

Figura 7. Motor de búsqueda

2.6 EVALUACIÓN DEL SISTEMA

En la evaluación del sistema de recuperación de imágenes, se utilizó el modelo VGG16 preentrenado en ImageNet, omitiendo la capa superior de clasificación, junto con el algoritmo KNN configurado con diferentes parámetros. A continuación, se presenta un análisis detallado de los resultados obtenidos para cada configuración.

Algoritmo: brute con métrica cosine

En primer lugar, se empleó el algoritmo brute junto con la métrica de distancia cosine. Esta configuración resultó en una precisión promedio de 0.5486 y un recall promedio de 0.4935. Estos resultados indican que esta combinación de parámetros es efectiva para capturar la similitud entre las imágenes en el espacio de características generado por VGG16. La distancia promedio observada fue de 0.1246, lo que sugiere que las imágenes recuperadas estaban bastante cercanas en términos de similitud. No obstante, el tiempo de recuperación promedio fue de 0.5510 segundos, el cual fue el más alto entre todas las configuraciones evaluadas. Aunque esta combinación proporciona la mayor precisión y recall, el costo en términos de tiempo de recuperación es considerablemente mayor.

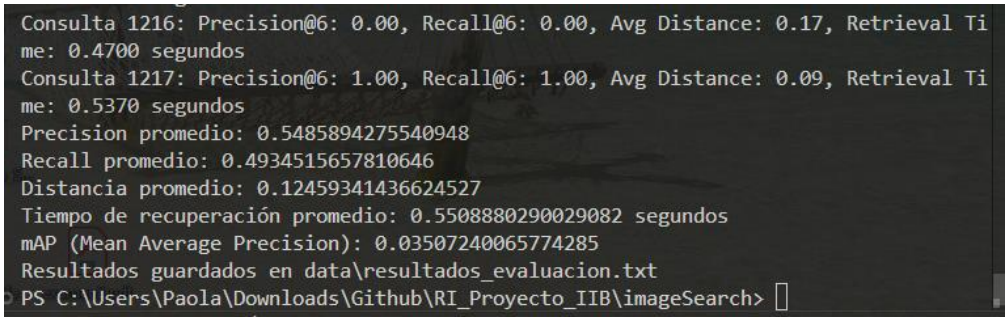

Figura 8. Evaluación con el algoritmo Brute

Algoritmo: ball_tree

El siguiente conjunto de parámetros evaluado fue el algoritmo ball_tree. Los resultados obtenidos con esta configuración mostraron una precisión promedio de 0.5056 y un recall promedio de 0.4493, ambos ligeramente inferiores en comparación con el algoritmo brute. La distancia promedio fue significativamente mayor, alcanzando 44.8440, lo que indica que las imágenes recuperadas estaban más dispersas en el espacio de características. Sin embargo, el tiempo de recuperación promedio se redujo notablemente a 0.1427 segundos, lo que hace que esta configuración sea mucho más rápida. A pesar de la disminución en precisión y recall, el menor tiempo de recuperación podría ser ventajoso en aplicaciones donde la velocidad es crucial.

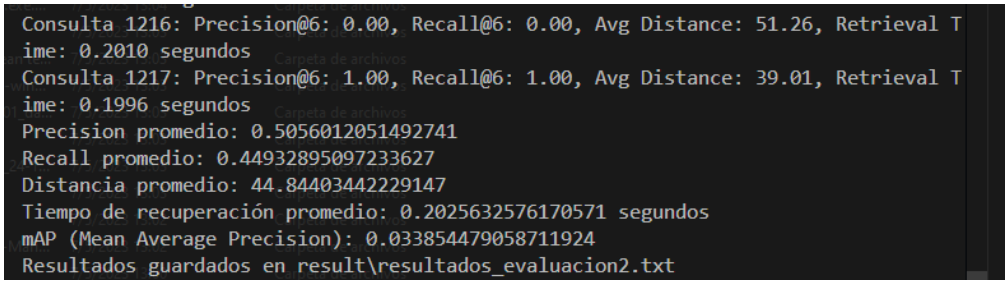

Figura 9. Evaluación con el algoritmo Ball tree

Algoritmo: kd_tree

Finalmente, se evaluó el rendimiento del sistema utilizando el algoritmo kd_tree. Los resultados obtenidos fueron similares a los del ball_tree, con una precisión promedio de 0.5056 y un recall promedio de 0.4493. La distancia promedio se mantuvo alta, en 44.8440, lo que sugiere que kd_tree no ofrece ventajas significativas en términos de precisión respecto a ball_tree. El tiempo de recuperación promedio fue de 0.2026 segundos, situándose entre los tiempos observados para brute y ball_tree.

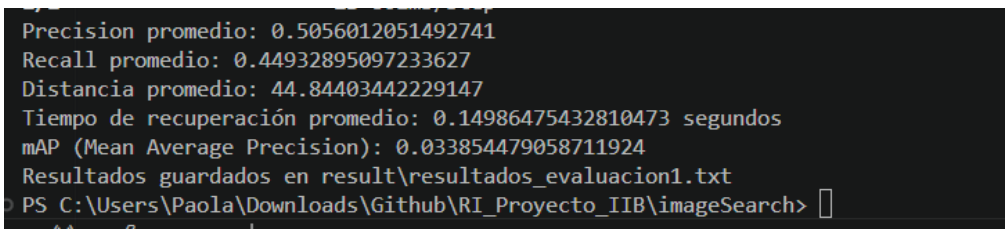

Figura 10. Evaluación con el algoritmo KD tree

En conclusión, el algoritmo brute con la métrica cosine proporciona la mayor precisión y recall, lo que lo convierte en la opción más adecuada cuando la exactitud del sistema es la prioridad. Sin embargo, esto viene a expensas de un mayor tiempo de recuperación. Por otro lado, los algoritmos ball_tree y kd_tree ofrecen tiempos de recuperación considerablemente más bajos, aunque con una precisión y recall ligeramente inferiores y con distancias promedio más altas, lo que podría afectar la relevancia de las imágenes recuperadas.

El análisis sugiere que, aunque brute con cosine es la configuración más precisa, podría ser necesario explorar otras métricas de distancia o incluso un enfoque híbrido para optimizar el rendimiento general del sistema.

2.7 INTERFAZ WEB DEL USUARIO

El objetivo de la interfaz web para el sistema FindMyPic es proporcionar una plataforma intuitiva y eficaz para la carga y búsqueda de imágenes similares. Esta interfaz está diseñada para ser accesible y fácil de usar, facilitando a los usuarios la interacción con el sistema de recuperación de imágenes mediante una experiencia fluida y visualmente atractiva.

La interfaz web se compone de dos páginas principales: la página de carga de imágenes y la página de resultados. Ambas páginas están diseñadas utilizando Tailwind CSS, lo que permite una apariencia moderna y uniforme a través de una paleta de colores degradados y un diseño centrado.

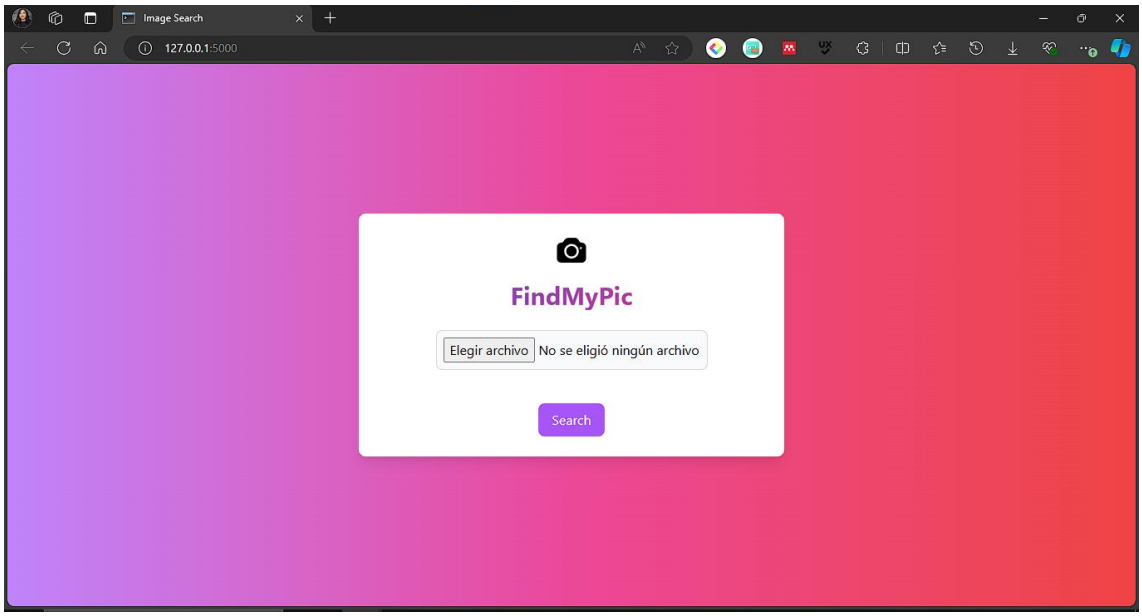

Figura11. Interfaz web

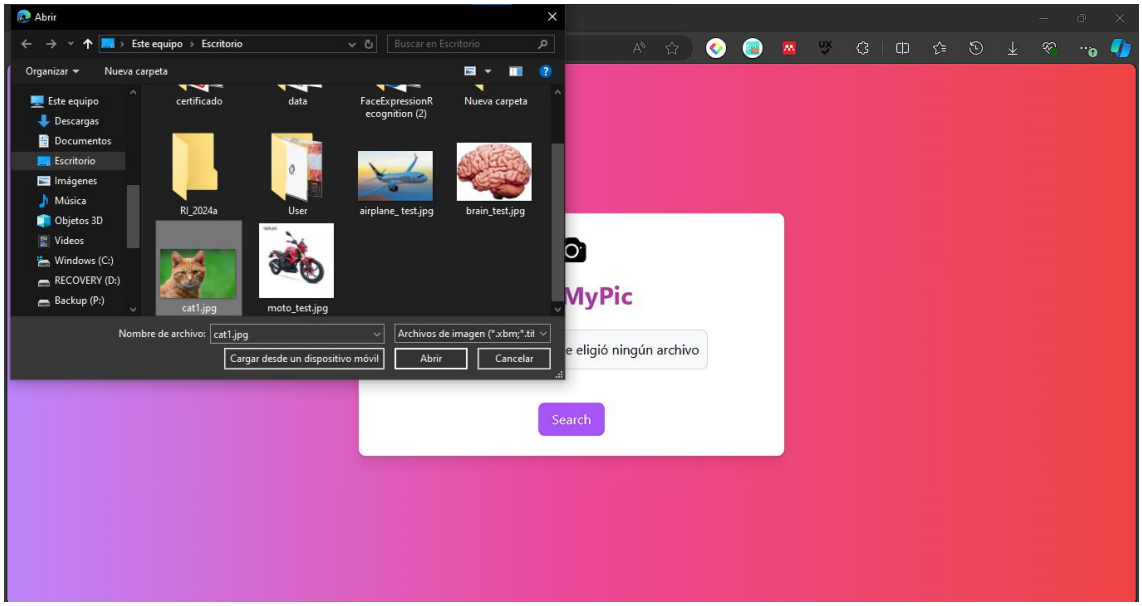

Figura 12. Interfaz web-carga de imagen

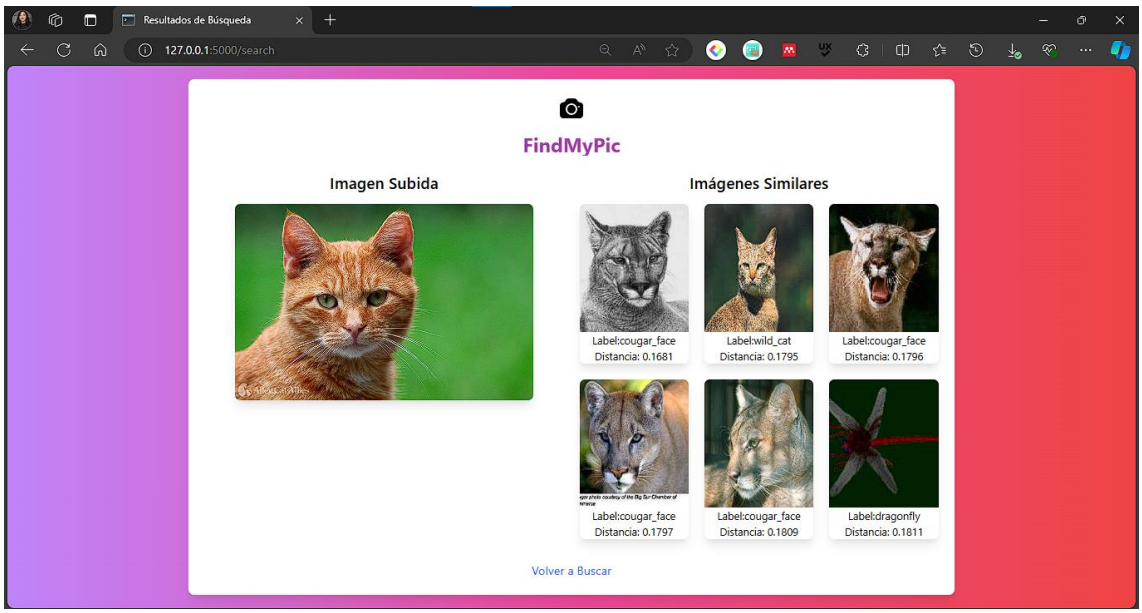

Figura 13. Interfaz web-resultados

3. REPOSITORIO DEL PROYECTO

El código fuente y los recursos asociados al proyecto FindMyPic están disponibles en el repositorio de GitHub. El repositorio incluye todos los archivos necesarios para la implementación y ejecución del sistema, así como la documentación adicional.

https://github.com/PaolaMaribel18/RI_Proyecto_IIB/tree/main