Proyecto Bimestral: Sistema de Recuperación de Información basado en Imágenes

Prof. Iván Carrera

30 de julio de 2024

Nombres: Kevin Mantilla, Carlos Sánchez

Fecha: 12/08/2024

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 dividirla en varias fases, que se describen a continuación.

2. Entrega Final

Documentación Completa: Incluyendo los procesos, decisiones tomadas, y resultados de cada fase.

Backend:

- **Inicialización del Sistema:** Se utilizó el modelo preentrenado de InceptionV3 para la extracción de características y un modelo de vecinos más cercanos (nearest_neighbors_model.pkl) para la búsqueda de imágenes similares. La base de datos MongoDB se utilizó para almacenar y recuperar imágenes.
- **Preprocesamiento de Imágenes:** Las imágenes se redimensionan a 224x224 píxeles y se normalizan para que los valores de los píxeles estén entre 0 y 1, lo cual es esencial para la entrada en modelos de aprendizaje profundo.
- **Predicción y Recuperación de Imágenes:** A través de una API, el sistema recibe una imagen, extrae sus características utilizando InceptionV3, y luego realiza una búsqueda en la base de datos para encontrar imágenes similares.

Frontend:

- **Configuración del Proyecto:** El frontend está construido con Next.js, utilizando React y TypeScript para un desarrollo moderno y eficiente. Tailwind CSS es utilizado para el diseño de la interfaz.
- **Manejo de Dependencias:** Las dependencias clave incluyen Axios para manejar solicitudes HTTP y la configuración estricta de TypeScript para mantener un código robusto y libre de errores.

Código Fuente: Organizado y bien comentado.

Comenzamos con el Preprocesamiento:

```python
# Cargar el dataset de Caltech101 desde TFDS
(train_dataset, test_dataset), dataset_info = tfds.load(
    name='caltech101',
    split=['train[:80%]','test[:90%]'],
    with_info=True,
    as_supervised=True,
    data_dir='',
    download=False
)
```

- Carga el dataset Caltech101 desde tensorflow_datasets, separándolo en conjuntos de entrenamiento (80% de los datos) y prueba (90% de los datos).

```python
# numero de clases
num_classes = dataset_info.features['label'].num_classes
print('Numero de clases:', num_classes)
```

[3] ✓ 0.0s

```
Numero de clases: 102
```

```python
# numero de imagenes
num_train_examples = dataset_info.splits['train'].num_examples
num_test_examples = dataset_info.splits['test'].num_examples
print('Numero de imagenes de entrenamiento:', num_train_examples)
print('Numero de imagenes de prueba:', num_test_examples)
```

[4] ✓ 0.0s

```
Numero de imagenes de entrenamiento: 3059
Numero de imagenes de prueba: 6085
```

- La primera celda obtiene y muestra el número de clases en el dataset.
- La segunda muestra la cantidad de imágenes en los conjuntos de entrenamiento y prueba.

preprocesamiento

```python
# Fijar la semilla para la reproducibilidad
tf.random.set_seed(1234)
np.random.seed(1234)
```

[5] ✓ 0.0s

```python
# Función para preprocesar las imágenes
def preprocess_image(image, label):
    image = tf.image.resize(image, (224, 224))
    image = tf.cast(image, tf.float32) / 255.0
    return image, label
```

[6] ✓ 0.0s

```python
# Aplicar el preprocesamiento
train_dataset_process = train_dataset.map(preprocess_image).shuffle(1000).batch(32)
test_dataset_process = test_dataset.map(preprocess_image).batch(32)
```

[7] ✓ 0.0s

- En la segunda celda la función redimensiona las imágenes a 224x224 píxeles y las normaliza para que los valores de los píxeles estén en el rango [0, 1].
- En la tercera celda se aplica el preprocesamiento al dataset y organiza los datos en lotes (batchs) para el entrenamiento y prueba.

Extraccion de caracteristicas

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

#limpiar memoria
del base_model
```

[8] ✓ 2.0s

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

[9] ✓ 0.0s

```
# Extraer características para el conjunto de entrenamiento
train_features, train_labels, train_images = extract_features(train_dataset_process)
```

[10] ✓ 53.4s

- Este código carga un modelo preentrenado InceptionV3 sin su capa final de clasificación para utilizarlo como extractor de características en imágenes. Luego define una función que itera sobre un conjunto de datos de imágenes, aplicando el modelo para extraer características de cada imagen, y almacenando tanto estas características como las etiquetas y las imágenes originales en listas. Finalmente, se aplica esta función al conjunto de datos de entrenamiento para obtener las características que se utilizarán en análisis posteriores.

```
# Aplanar los mapas de características
train_features_flat = np.array([feature.flatten() for batch in train_features for feature in batch])
# Aplanar las etiquetas y los nombres de archivo
train_labels_flat = np.array([label for batch in train_labels for label in batch])
train_images_flat = np.array([fl for batch in train_images for fl in batch])
```

[11] ✓ 1.2s

```
# Guardar las características en un npz
np.savez_compressed('caltech101_train_features.npz', features=train_features_flat, labels=train_labels_flat, Images=train_images_flat)


# limpiar memoria
del train_features, train_labels, train_images, train_features_flat, train_labels_flat, train_images_flat
```

[12] ✓ 2m 1.1s

- Este fragmento de código aplana los mapas de características extraídos, así como las etiquetas y las imágenes originales en arreglos de NumPy, para luego guardar estos datos en un archivo comprimido. npz llamado caltech101_train_features.npz. Finalmente, se eliminan de la memoria las variables utilizadas para liberar espacio.

- Este código carga un archivo .npz que contiene características previamente extraídas, etiquetas e imágenes, y luego ajusta un modelo de NearestNeighbors con 5 vecinos utilizando el algoritmo ball_tree para indexar las características. Finalmente, el modelo entrenado se guarda en un archivo pkl usando joblib, para poder reutilizarlo en futuras predicciones o análisis sin necesidad de reentrenar el modelo.

Backend:

- La API Contiene la lógica principal del servidor, manejo de la API, y conexión a la base de datos.

Primero importamos todas las librerías:

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from joblib import load
import numpy as np
from PIL import Image
import io
import base64
import tensorflow as tf
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Model
from pymongo import MongoClient
```

A continuación, se explicará función del código:

```
app = Flask(__name__)
CORS(app)
```

- Flask(__name__): Crea una instancia de la aplicación Flask.
- CORS(app): Habilita el soporte para solicitudes CORS (Cross-Origin Resource Sharing), permitiendo que la aplicación frontend interactúe con este backend desde diferentes dominios.

Luego se definen las variables globales:

```python
# Variables globales para modelos y datos
model = None
model_nn = None
collection = None
```

- model: El modelo de extracción de características basado en InceptionV3.
- model_nn: El modelo de vecinos más cercanos (K-Nearest Neighbors) utilizado para buscar imágenes similares.
- collection: La colección de imágenes en MongoDB.

```python
def initialize():
    global model, model_nn, collection
    # Cargar el modelo de características
    base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    model = Model(inputs=base_model.input, outputs=base_model.layers[-1].output)

    # Limpiar memoria
    del base_model
    tf.keras.backend.clear_session()

    # Cargar el modelo de vecinos más cercanos
    model_nn = load('nearest_neighbors_model.pkl')

    # Conectar a MongoDB
    client = MongoClient('mongodb://localhost:27017/')
    db = client['caltech101db']
    collection = db['images']
```

- base_model: Carga InceptionV3 preentrenado sin la capa de clasificación superior (solo capas convolucionales) para extraer características de las imágenes.
- model: Extrae la salida de la última capa convolucional del modelo base.
- Limpieza de memoria: Después de crear el modelo, se elimina base_model y se limpia la memoria de TensorFlow.
- model_nn: Carga un modelo de vecinos más cercanos (previamente entrenado) que permite encontrar imágenes similares.
- MongoDB: Se conecta a la base de datos MongoDB y se accede a la colección de imágenes.

Ahora definimos la siguientes funciones:

```python
def preprocess_image(image):
    image = tf.image.resize(image, (224, 224))  # Redimensiona la imagen
    image = tf.cast(image, tf.float32) / 255.0  # Normaliza la imagen
    return image

def image_to_base64(image):
    buffered = io.BytesIO()
    image.save(buffered, format="PNG")
    return base64.b64encode(buffered.getvalue()).decode("utf-8")

def get_image_from_db(index):
    image_doc = collection.find_one({'index': index}, {'_id': 0, 'image': 1})
    if image_doc:
        return Image.open(io.BytesIO(image_doc['image']))
    return None
```

- Redimensionamiento: Redimensiona la imagen a 224x224 píxeles para que coincida con la entrada esperada por InceptionV3.

- Normalización: Convierte los valores de los píxeles a un rango de 0 a 1.
- Convierte una imagen de PIL en un string codificado en Base64, que se puede enviar a través de JSON para mostrar en el frontend.
- Recupera una imagen desde MongoDB dado su índice y la convierte en un objeto de imagen de PIL.

Luego realizamos lo siguiente:

```python
@app.route('/api/predict', methods=['POST'])
def predict():
    if 'image' not in request.files:
        return jsonify({'error': 'No image file provided'}), 400

    try:
        image = request.files['image']
        img = Image.open(image)
        img_uploaded_base64 = image_to_base64(img)

        img_preprocessed = preprocess_image(tf.convert_to_tensor(np.array(img)))
        img_preprocessed = tf.expand_dims(img_preprocessed, axis=0)

        query_features = model.predict(img_preprocessed).flatten().reshape(1, -1)

        distances, indices = model_nn.kneighbors(query_features, n_neighbors=10)
        # print(indices[0])

        closest_images = [image_to_base64(get_image_from_db(int(index))) for index in indices[0] if get_image_fr

        return jsonify({
            'closest_images': closest_images,
            'uploaded_image': img_uploaded_base64
        })

    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

- Ruta /api/predict: Se define una API POST que permite subir una imagen y recibir las imágenes más similares de la base de datos.
- Entrada de Imagen: Verifica que se haya subido una imagen en la solicitud.
- Procesamiento de la Imagen: La imagen se procesa (redimensiona y normaliza) y se extraen sus características usando el modelo InceptionV3.
- Predicción: Utiliza el modelo de vecinos más cercanos para encontrar las 10 imágenes más similares en la base de datos.
- Salida: Devuelve las imágenes más similares (codificadas en Base64) y la imagen cargada por el usuario.

Por último, se inicia la aplicación Flask en modo de depuración en el puerto 8000.

```python
if __name__ == '__main__':
    app.run(debug=True, port=8000)
```

- Para la base de datos se realiza el siguiente proceso:

```python
# Configuración de la aplicación Flask
app = Flask(__name__)
app.config['MONGO_URI'] = 'mongodb://localhost:27017/caltech101db'
mongo = PyMongo(app)
```

- Configuración de MongoDB: Define la URI de la base de datos MongoDB donde se almacenarán las imágenes.

```python
def save_image_as_bytes(image_array):
    # Convertir el array NumPy a imagen PIL
    image_pil = Image.fromarray((image_array * 255).astype('uint8'))

    # Guardar la imagen en un buffer
    buffer = io.BytesIO()
    image_pil.save(buffer, format='PNG')
    return buffer.getvalue()
```

- Propósito: Convierte un array NumPy (que representa una imagen) en una imagen de PIL y luego la guarda en un stream de bytes.
- Uso: Los bytes resultantes se almacenan en MongoDB.

```python
def populate_db():
    # Conectar a la colección en MongoDB
    images_collection = mongo.db.images

    # Limpiar la colección antes de poblarla
    images_collection.delete_many({})

    try:
        # Cargar las imágenes y etiquetas desde el archivo .npz
        data = np.load('caltech101_train_features.npz')
    except Exception as e:
        print(f"Error al cargar el archivo .npz: {e}")
        return

    # Verificar si las claves existen en el archivo .npz
    if 'labels' not in data or 'Images' not in data:
        print("El archivo .npz no contiene las claves 'labels' o 'Images'")
        return

    train_labels_loaded = data['labels']
    train_images_loaded = data['Images']

    # Verificar que la cantidad de imágenes y etiquetas coincide
    if len(train_images_loaded) != len(train_labels_loaded):
        print("La cantidad de imágenes y etiquetas no coincide")
        return
```

```python
# Poblar la base de datos con las imágenes de entrenamiento
for i in range(len(train_images_loaded)):
    image_array = train_images_loaded[i]
    label = str(train_labels_loaded[i])

    # Convertir la imagen a bytes
    image_bytes = save_image_as_bytes(image_array)

    # Insertar el documento en MongoDB
    document = {
        'image': image_bytes,
        'label': label,
        'index': i
    }

    try:
        images_collection.insert_one(document)
    except Exception as e:
        print(f"Error al insertar el documento en MongoDB: {e}")
        return

print("Base de datos poblada con éxito")
```

- Limpieza de la colección: Se eliminan todos los documentos anteriores en la colección antes de añadir nuevos.
- Carga de Datos: Carga las imágenes y etiquetas desde un archivo .npz.
- Verificación: Asegura que las imágenes y etiquetas coincidan en cantidad.
- Inserción en la Base de Datos: Convierte cada imagen a bytes y la inserta en MongoDB junto con su etiqueta y un índice.

```python
if __name__ == "__main__":
    with app.app_context():
        populate_db()
```

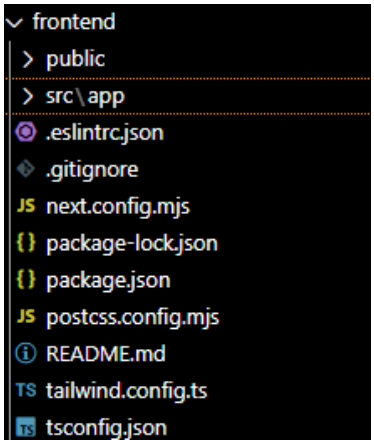
- Propósito: Ejecutar la función populate_db() dentro del contexto de la aplicación Flask, asegurando que la base de datos se pueble cuando se ejecute el script.

```python
# Conectar a MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['caltech101db']
collection = db['images']

# Cargar los datos desde el archivo NPZ
data = np.load('caltech101_train_features.npz')
npz_images = data['Images']

# Comparar las imágenes en NPZ y MongoDB
for index in range(len(npz_images)):
    # Obtener imagen de NPZ
    npz_image_array = npz_images[index]
    npz_image = Image.fromarray((npz_image_array * 255).astype('uint8'))

    # Obtener imagen de MongoDB
    image_doc = collection.find_one({'index': index})
    if image_doc:
        mongo_image = Image.open(io.BytesIO(image_doc['image']))

        # Comparar las dos imágenes
        if np.array_equal(np.array(npz_image), np.array(mongo_image)):
            print(f"Image at index {index} matches")
        else:
            print(f"Image at index {index} does not match")
    else:
        print(f"No image found in MongoDB at index {index}")
```

- MongoClient: Establece una conexión con la base de datos MongoDB que se está ejecutando en la máquina local (localhost) en el puerto 27017, que es el puerto por defecto de MongoDB.
- db: Accede a la base de datos llamada caltech101db.
- collection: Accede a la colección images dentro de la base de datos, donde se almacenan las imágenes.
- np.load: Carga los datos desde el archivo .npz, que contiene las imágenes en forma de arrays NumPy.
- npz_images: Extrae los arrays de imágenes desde el archivo .npz y los guarda en la variable npz_images.
- npz_image_array: Extrae el array de la imagen en la posición index.
- npz_image: Convierte el array en una imagen de PIL. Multiplica por 255 para restaurar los valores de píxel al rango original [0, 255] (asumiendo que originalmente fueron normalizados entre 0 y 1).
- collection.find_one({'index': index}): Busca en la colección de MongoDB la imagen que tiene el mismo índice.
- mongo_image: Si encuentra un documento con ese índice, lo convierte en una imagen de PIL usando los datos almacenados en MongoDB.
- np.array_equal(np.array(npz_image), np.array(mongo_image)): Compara la imagen obtenida del archivo .npz con la imagen obtenida de MongoDB, pixel por pixel.

Frontend:

- El código fuente está organizado en carpetas clave como components/ para componentes reutilizables y public/ para recursos estáticos, lo que facilita la navegación y el mantenimiento.
- **Archivos de Configuración:** Los archivos next.config.mjs y tsconfig.json están configurados para optimizar el entorno de desarrollo, garantizando un desarrollo eficiente y estructurado.



Informe de Evaluación: Análisis detallado de la evaluación del sistema.

Por último, vamos a realizar las métricas precisión y recall utilizando sklearn.metrics import classification_report,accuracy_score, precision_score, recall_score, f1_score:

```python
# precision, recall, f1-score
from sklearn.metrics import classification_report,accuracy_score, precision_score, recall_score, f1_score

# Obtener las etiquetas predichas
predicted_labels = [train_labels_loaded[idx] for idx in indices_test[:,0]]

# Generar el reporte de clasificación
report = classification_report(test_labels_flat, predicted_labels)

# Imprimir el reporte
print(report)
```

Python

Compara las etiquetas reales del conjunto de prueba con las etiquetas predichas por el modelo y genera un informe de clasificación detallado. Este informe muestra las métricas de rendimiento para cada clase y también proporciona un resumen general del rendimiento del modelo en términos de precisión, recall, y F1-score. Esto es muy útil para evaluar qué tan bien está funcionando el modelo en diferentes categorías o clases.

```python
accuracy = accuracy_score(test_labels_flat, predicted_labels)
precision = precision_score(test_labels_flat, predicted_labels, average='macro')
recall = recall_score(test_labels_flat, predicted_labels, average='macro')
f1 = f1_score(test_labels_flat, predicted_labels, average='macro')

print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision * 100:.2f}%")
print(f"Recall: {recall * 100:.2f}%")
print(f"F1 Score: {f1 * 100:.2f}%")
```

[37] Python

```
Accuracy: 78.56%
Precision: 77.79%
Recall: 78.64%
F1 Score: 74.38%
```

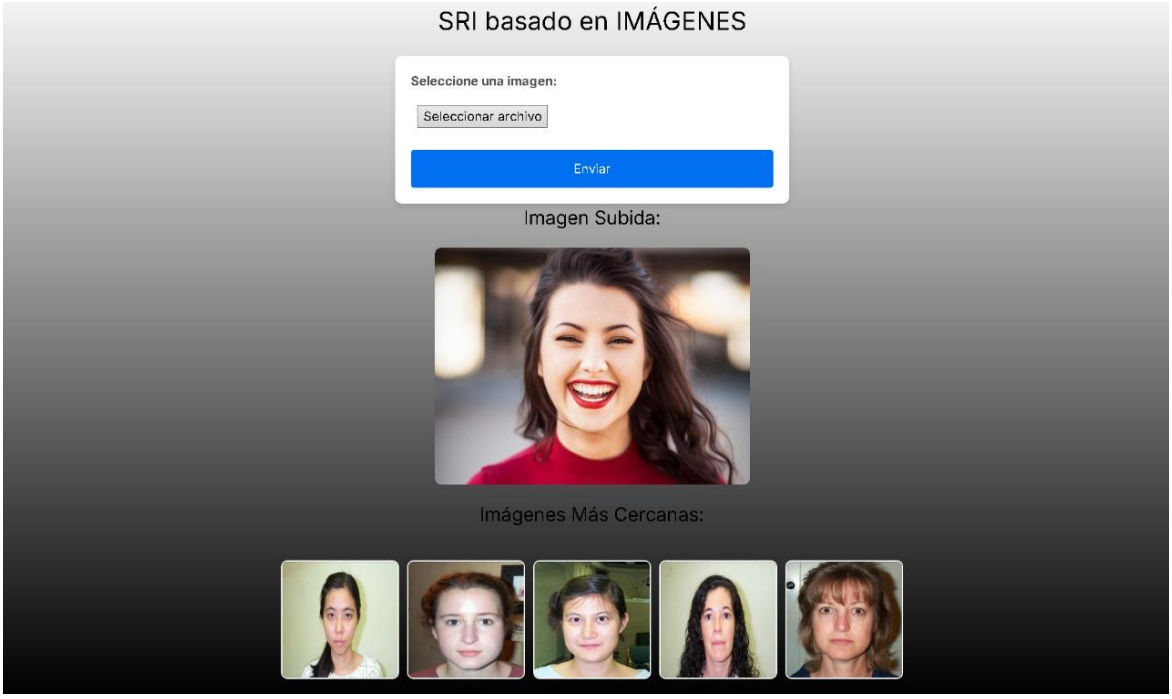
Evalúa el rendimiento de un modelo de clasificación utilizando cuatro métricas clave: exactitud (accuracy), precisión (precision), recall (sensibilidad). Los resultados de estas métricas se muestran como porcentajes con dos decimales, lo que facilita la interpretación del rendimiento del modelo.

Demostración del Sistema: Presentación funcional del sistema a través de la interfaz web.

Primero tenemos la interfaz normal de la siguiente manera:

Ahora una vez ingresada una imagen el sistema busca los 5 resultados más cercanos.

Ahora otro ejemplo con una imagen diferente:



Repositorio de GitHub: https://github.com/CarlosxSL/Image-based-Information-Retrieval-System.git