ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS
INGENIERÍA EN COMPUTACIÓN

Proyecto Bimestral: Sistema de Recuperación de Información basado en Imágenes

Integrantes:

- Fernández Achig Madelyn Angelyca
- Valle Morales Kevin Esteban

Fecha de entrega: 13/08/2024

Prof. Iván Carrera
30 de julio de 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 encontrar imágenes similares en una base de datos dada. El proyecto se dividirá en varias en varias fases, que se describen a continuación.

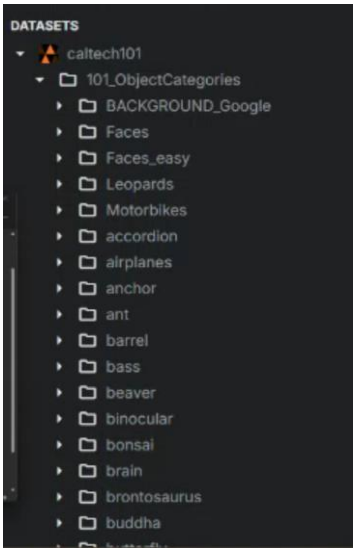
2. Fases del Proyecto

2.1. Adquisición de Datos

Objetivo: Obtener y preparar el dataset Caltech101.

Tareas:

- Descargar el dataset.
- Descomprimir y organizar los archivos.



2.2. Preprocesamiento

Objetivo: Preparar los datos para su análisis.

Tareas:

- Usar técnicas de preprocesamiento de imagenes, como normalización, reducción de tamaño, o eliminación de ruido.
- Documentar cada paso del preprocesamiento.

configura un flujo de trabajo para procesar imágenes desde un directorio, aplicarles un preprocesamiento, y pasarlas a través de un modelo para obtener sus características.

```
# Tamaño del lote de imágenes a procesar en cada iteración
batch_size = 64

# Directorio raíz donde se encuentran las imágenes organizadas por categorías
root_dir = '/kaggle/input/caltech101/101_ObjectCategories'

# Crear un generador de imágenes que aplicará la función de preprocesamiento necesaria para el modelo
img_gen = ImageDataGenerator(preprocessing_function=preprocess_input)

# Configurar el generador para leer imágenes desde el directorio raíz, sin clasificación de categorías y sin baraja
datagen = img_gen.flow_from_directory(root_dir,
                                      target_size=(img_size, img_size),  # Tamaño al que se redimensionarán las
                                      batch_size=batch_size,             # Número de imágenes por lote
                                      class_mode=None,                   # No se necesitan etiquetas de clases
                                      shuffle=False)                     # Procesar imágenes en orden de directo

# Calcular el número total de imágenes en el directorio
num_images = len(datagen.filenames)

# Calcular el número de iteraciones necesarias para procesar todas las imágenes una vez
```

2.3. Extracción de Características

Objetivo: Extraer las características de las imágenes en una forma que los algoritmos puedan procesar.

Tareas:

- Utilizar un modelo de red neuronal convolucional (CNN) para extraer características de las imágenes.
- Entrenar un modelo desde cero o utilizar un modelo preentrenado y aplicar transfer learning.
- Documentar los métodos y resultados obtenidos.

La extracción de características se llevó a cabo utilizando ResNet, un modelo de CNN que ha demostrado un alto rendimiento en tareas de reconocimiento de imágenes. ResNet fue elegido por su capacidad para manejar la complejidad de imágenes y por su arquitectura profunda, que incluye bloques residuales que permiten la conservación de la información a través de muchas capas. Se utilizó la versión preentrenada de ResNet, entrenada en el dataset ImageNet, lo que facilitó el proceso de transfer learning. Se realizaron ajustes menores para adaptar el modelo a las especificidades del dataset utilizado en este proyecto.

```
model = ResNet50(weights='imagenet', include_top=False,input_shape=(img_size, img_size, 3),pooling='max')
```

Imágenes clasificadas por carpetas de clase, carga cada imagen, procesa y extrae sus características usando un modelo pre-entrenado.

[4]:

```
# Diccionario para almacenar las características extraídas de cada imagen
features_data = {}

# Recorrer cada clase (carpeta) en el directorio raíz
for class_name in os.listdir(root_dir):
    class_dir = os.path.join(root_dir, class_name)  # Construir la ruta completa al directorio de la clase

    # Asegurarse de que el camino es un directorio antes de proceder
    if os.path.isdir(class_dir):
        # Recorrer cada imagen en el directorio de la clase
        for img_name in os.listdir(class_dir):
            img_path = os.path.join(class_dir, img_name)  # Construir la ruta completa a la imagen

            # Verificar que el camino sea un archivo para evitar procesar entidades que no son imágenes
            if os.path.isfile(img_path):
                # Cargar la imagen, redimensionándola al tamaño requerido por el modelo
                img = image.load_img(img_path, target_size=(224, 224))
                img_data = image.img_to_array(img)  # Convertir la imagen cargada a un arreglo
                img_data = np.expand_dims(img_data, axis=0)  # Añadir una dimensión al arreglo para procesamiento
                img_data = preprocess_input(img_data)  # Aplicar preprocesamiento requerido por el modelo
```

Después de la extracción, las características obtenidas de las imágenes fueron de alta dimensionalidad, capturando detalles que son esenciales para la diferenciación de imágenes similares. Estas características fueron guardadas en un archivo. pkl, que proporciona una forma eficiente de almacenamiento y recuperación para su posterior uso en el proceso de búsqueda. La documentación incluye ejemplos de las características extraídas y su interpretación en el contexto de la búsqueda de imágenes.

```
# Guardar las características extraídas en un archivo .pkl
with open('caltech101_features_new.pkl', 'wb') as f:
    pickle.dump(features_data, f)
```

2.4 Indexación

Objetivo: Crear un índice que permita búsquedas eficientes.

Tareas:

- Desarrollar un sistema para indexar las características extraídas de las imágenes y permitan la búsqueda eficiente de imágenes similares.
- Considerar técnicas como k-NN o índices especializados como KD-Trees o LSH (LocalitySensitive Hashing).
- Documentar el proceso de construcción del índice.

A continuación, se observa una parte de las características obtenidas anteriormente, las cuales van a ser utilizadas con el algoritmo kNN.

```
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0073.jpg : [ 2.3513677 10.287294   4.151931  ...
6.0459175 15.684694   0.       ]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0040.jpg : [1.6256844 3.4343762 5.3545094 ... 1.23
46523 6.3165884 4.783993 ]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0030.jpg : [11.483638    2.4619372  16.645752
...  2.8895128   1.9200833
  0.34722292]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0062.jpg : [10.268625   0.6232114 11.056247  ...
3.496349   1.9965128  3.1374097]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0020.jpg : [ 4.186825   3.262821  10.563641  ... 1
1.38591    0.7677895  0.       ]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0044.jpg : [1.10709524e+01 2.95054674e+00 1.344801
24e+01 ... 7.34138012e+00
 1.42880428e+00 1.04750395e-02]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0023.jpg : [ 3.7262497  4.8760443  4.1353683 ...
9.047211   3.173867  21.281582 ]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0050.jpg : [ 8.754116   4.7282953  9.232899  ... 1
0.502331   5.204416   6.554863 ]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0041.jpg : [ 4.9308963  9.23581   12.790461  ...
6.503357   2.1813283  3.2287664]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0047.jpg : [5.159033  2.2889771 8.028842  ... 1.70
05951 3.3144503 1.9641923]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0070.jpg : [ 5.655602  11.318088  15.234875  ... 1
1.313262   1.0660306  6.967557 ]
Features for /kaggle/input/caltech101/101_ObjectCategories/scorpion/image_0036.jpg : [6.298073   6.0436664  6.9662538  ...
9.220945   0.30028808 0.26835802]
```

La indexación de las características se realizó mediante el uso del algoritmo k-Nearest Neighbors (k-NN), que es particularmente adecuado para tareas de clasificación y búsqueda en grandes conjuntos de datos de alta dimensionalidad. La implementación de k-NN se optimizó para manejar eficientemente las consultas en un dataset extenso, lo que implica calcular la distancia entre las características de una imagen dada y todas las imágenes indexadas.

2.5. Diseño del Motor de Búsqueda

Objetivo: Implementar la funcionalidad de búsqueda.

Tareas:

- Desarrollar la lógica para procesar consultas de usuarios a partir de imágenes.
- Desarrollar un algoritmo de ranking para ordenar los resultados.
- Documentar la arquitectura y los algoritmos utilizados.

La lógica del motor de búsqueda se basa en la extracción de características de una imagen de consulta utilizando el modelo ResNet, seguida por la comparación de estas características con las ya indexadas. El algoritmo k-NN se encarga de identificar las imágenes más similares, calculando la distancia euclidiana entre los vectores de características. Los resultados se ordenan según esta distancia, así que las imágenes más similares aparecen en las primeras posiciones del ranking.

Se diseñó una arquitectura modular para el motor de búsqueda, donde cada componente (extracción, indexación, búsqueda y ranking) se implementó de manera independiente para facilitar futuras mejoras y mantenimiento.

Primero tenemos la carga del modelo ResNet50, se establece la ruta del dataset y cargamos el archivo pkl con las características obtenidas previamente.

```
# Cargar el modelo ResNet50 preentrenado en ImageNet
model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3), pooling='max')

# Ruta al dataset de Caltech101
dataset_path = 'D:/Kevin/Documents/EPN/2024A/Recuperación de Información/Proyecto-RI-IIB-2024A/data_caltech/101_ObjectCategories'

# Cargar las características de las imágenes desde el archivo .pkl
with open('D:/Kevin/Documents/EPN/2024A/Recuperación de Información/Proyecto-RI-IIB-2024A/backend/image-upload-and-predict/model/caltech101_feature
    image_paths = pickle.load(f)
```

Se define la función que va a permitir extraer las características en este caso para la imagen de consulta, luego se extrae las características del dataset para entrenar el modelo kNN, la métrica utilizada es la distancia euclidiana.

```
# Función para extraer características usando ResNet50
def extract_features(img_array):
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)
    features = model.predict(img_array)
    return features.flatten()

# Extraer características para todas las imágenes
X = np.array(list(image_paths.values()))
y = [path.split('\\')[-2] for path in image_paths]  # Extraer la etiqueta del path

# Entrenar el modelo KNN
knn = KNeighborsClassifier(n_neighbors=5, algorithm='brute', metric='euclidean')
knn.fit(X, y)
```

Luego se define dentro del endpoint upload con una petición POST la siguiente función, la cual se encarga de subir la imagen, procesarla, extraer las características y obtener las 5 mejores coincidencias usando kNN.

Para que el frontend pueda obtener las imágenes, estas se convierten o codifican a base64, se recorre con un ciclo for las características de las imágenes y por ultimo se devuelve la imagen de consulta, las 5 imágenes más similares con su respectivo índice y distancia calculada.

```python
def upload_image():
    if 'image' not in request.files:
        return jsonify({'error': 'No image uploaded'}), 400

    file = request.files['image']

    try:
        # Procesar la imagen
        img = Image.open(file).convert('RGB')
        img = img.resize((224, 224))
        img_array = np.array(img)

        # Extraer características de la imagen recibida
        features = extract_features(img_array)

        # Obtener las 5 mejores coincidencias usando KNN
        distances, indices = knn.kneighbors([features])
        best_matches = [list(image_paths.keys())[i] for i in indices[0]]

        # Convertir la imagen subida a base64
        buffered = BytesIO()
        img.save(buffered, format="JPEG")
        uploaded_image_base64 = base64.b64encode(buffered.getvalue()).decode('utf-8')
        uploaded_image_base64 = f"{uploaded_image_base64}"

        # Convertir las imágenes a base64 para enviarlas al frontend
        images_base64 = []
        matches_info = []
        for i, match in enumerate(best_matches):
            with open(match, "rb") as img_file:
                b64_string = base64.b64encode(img_file.read()).decode('utf-8')
                images_base64.append(f"{b64_string}")
                matches_info.append({
                    'image': f"{b64_string}",
                    'distance': float(distances[0][i]),  # Convertir a tipo float
                    'index': int(indices[0][i])  # Convertir a tipo int
```

```python
        return jsonify({
            'uploaded_image': uploaded_image_base64,  # Incluir la imagen subida
            'matches': matches_info
        }), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

En el frontend se configura para subir la imagen de consulta y el endpoint del backend donde se carga esta imagen y se realiza el proceso anteriormente explicado.

Se maneja posibles errores que se pueden dar al momento de subir la imagen en la página web.

```
    try {
      const response = await fetch('http://127.0.0.1:5000/upload', {
        method: 'POST',
        body: formData,
      });

      if (response.ok) {
        const data = await response.json();
        setUploadedImage(URL.createObjectURL(file));
        setSimilarImages(data.matches);
      } else {
        console.error('Error al subir la imagen:', response.statusText);
        alert('Error al subir la imagen');
      }
    } catch (error) {
      console.error('Error al enviar la solicitud:', error);
      alert('Error al enviar la solicitud');
    } finally {
      setLoading(false);
    }
  } else {
    alert('Solo se permiten archivos PNG o JPG');
  }
```

El siguiente código realiza la obtención de la imagen subida y las 5 más similares procesadas desde el backend, con sus respectivos índices y distancias.

```
    <Box sx={styles.uploadedImageContainer}>
      <Typography variant="h6" sx={styles.title}>Uploaded Image</Typography>
      <img src={uploadedImage} alt="Uploaded" style={styles.uploadedImage} />
    </Box>
)}
{similarImages.length > 0 && !loading && (
  <Box sx={styles.similarImagesContainer}>
    <Typography variant="h6" sx={styles.title}>Similar Images</Typography>
    {similarImages.map((match, index) => (
      <Box sx={styles.imageWrapper} key={index}>
        <img src={match.image} alt={`Match ${index + 1}`} style={styles.similarImage} />
        <Typography variant="body1" sx={styles.infoText}>
          Índice: {match.index}
        </Typography>
        <Typography variant="body1" sx={styles.infoText}>
          Distancia: {match.distance.toFixed(2)}
        </Typography>
      </Box>
    ))}
  </Box>
```

2.6. Evaluación del Sistema

Objetivo: Medir la efectividad del sistema.

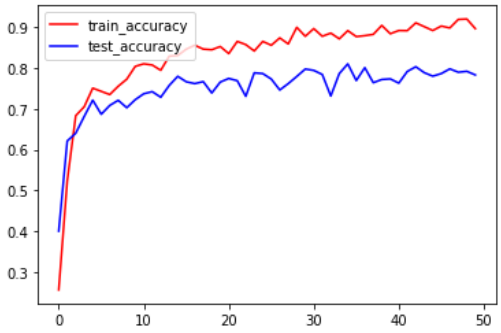
Tareas:

- Definir un conjunto de métricas de evaluación (precisión, recall, F1-score).
- Definir un benchmark para evaluación del sistema.
- Comparar el rendimiento de diferentes configuraciones del sistema.
- Documentar los resultados y análisis.

Para garantizar que el sistema cumple con los requisitos de precisión y eficiencia, se definieron métricas clave de evaluación, como la pérdida y la precisión categóricas, que proporcionaron una medida directa de la capacidad del sistema para clasificar correctamente las imágenes en función de su similitud.

Se obtuvo la siguiente grafica para el accuracy:

```
# Visualise train / Valid Accuracy
plt.plot(new_model.history.history["categorical_accuracy"], c="r", label="train_accuracy")
plt.plot(new_model.history.history["val_categorical_accuracy"], c="b", label="test_accuracy")
plt.legend(loc="upper left")
plt.show()
```

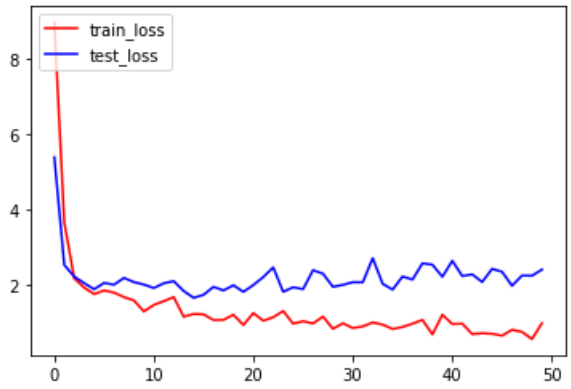


La precisión en el conjunto de entrenamiento muestra un crecimiento constante desde el inicio del entrenamiento, alcanzando un valor cercano a 0.9 hacia el final. Esto indica que el modelo fue capaz de aprender patrones en los datos de entrenamiento de manera efectiva, mejorando su rendimiento a medida que se realizaron más épocas.

La precisión en el conjunto de validación también mejora con el tiempo, aunque su crecimiento es más moderado en comparación con el conjunto de entrenamiento. La línea azul alcanza un valor cercano a 0.8, lo que sugiere que el modelo generaliza razonablemente bien a datos no vistos durante el entrenamiento, aunque su rendimiento es ligeramente inferior al observado en el conjunto de entrenamiento.

Y esta es la gráfica para la perdida:

```
# Visualise train / Valid Loss
plt.plot(new_model.history.history["loss"], c="r", label="train_loss")
plt.plot(new_model.history.history["val_loss"], c="b", label="test_loss")
plt.legend(loc="upper left")
plt.show()
```



La pérdida en el conjunto de entrenamiento disminuye rápidamente durante las primeras épocas, lo que es típico cuando el modelo comienza a aprender patrones en los datos. Posteriormente, la pérdida continúa disminuyendo de manera gradual, estabilizándose alrededor de un valor bajo. Esto indica que el modelo se está ajustando adecuadamente a los datos de entrenamiento.

La pérdida en el conjunto de validación también disminuye inicialmente, aunque no de forma tan pronunciada como en el conjunto de entrenamiento. A medida que avanzan las épocas, la pérdida de validación se estabiliza en un nivel más alto en comparación con la pérdida de entrenamiento y muestra una ligera tendencia a fluctuar, lo que es indicativo de **overfitting**.

2.7. Interfaz Web de Usuario

Objetivo: Crear una interfaz para interactuar con el sistema.

Tareas:

- Diseñar una interfaz web donde los usuarios puedan subir una imagen y recibir resultados con imágenes similares.
- Mostrar los resultados de búsqueda de manera clara y ordenada.
- La interfaz debe ser intuitiva y facilitar la interacción con el sistema.
- Documentar el diseño y funcionalidades de la interfaz.

Se diseñó y desarrolló una interfaz web utilizando **React** y **Material-UI** para ofrecer una experiencia de usuario moderna y responsiva. La interfaz permite a los usuarios cargar una imagen desde su dispositivo, que luego es procesada por el sistema para encontrar y

mostrar las imágenes más similares dentro de la base de datos. Los resultados de la búsqueda se presentan en un formato de galería, con las imágenes más similares ordenadas según su relevancia.

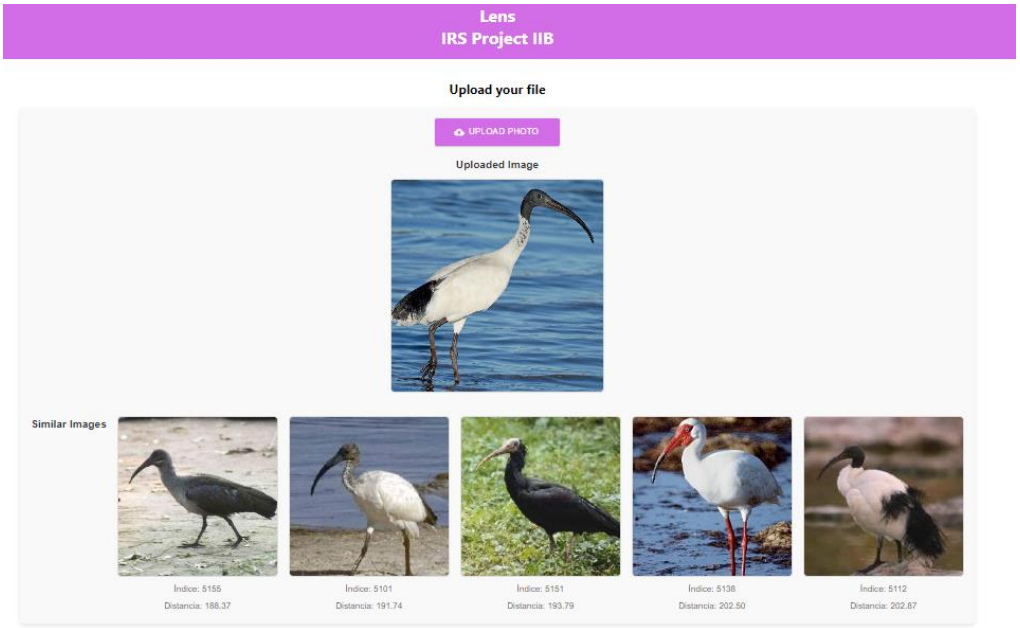
La interfaz se diseñó con un enfoque minimalista para garantizar que la atención del usuario se centrara en la funcionalidad principal. Los resultados de búsqueda se presentan con un alto grado de visualización, permitiendo a los usuarios ver cada imagen con claridad y comprender fácilmente la similitud entre las imágenes encontradas. Además, se incluyó la funcionalidad para imprimir los valores de índice y distancia de las imágenes similares, ofreciendo una visión más profunda de cómo el sistema determina la relevancia de los resultados.

La siguiente imagen nos muestra que al ingresar en la pagina se trata de un diseño minimalista y se centra en la función principal la cual es el subir o cargar una foto para la búsqueda de similitud.

Luego de cargar la imagen y de realizarse el procesamiento se obtiene una galeria, en la cual se muestra la imagen subida y las similitudes encontradas.

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS
INGENIERÍA EN COMPUTACIÓN



Diseño

1. **Color y Estética**:
    - La interfaz utiliza una combinación de colores suaves, predominando el lila en el botón y la barra superior, lo que proporciona una estética moderna y agradable.
    - Las imágenes están bien organizadas y centradas, lo que facilita la navegación visual.
2. **Disposición y Estructura**:
    - **Barra Superior**: Contiene el título de la aplicación, "Lens", que sugiere un enfoque en la visión o la fotografía.
    - **Botón de Carga**: Ubicado en la parte superior, permite al usuario cargar una imagen para su análisis o búsqueda de similitudes.
    - **Imagen Subida**: En el centro, se muestra la imagen que el usuario ha subido, resaltando el objeto de interés (en este caso, un ave ibis).
    - **Sección de Imágenes Similares**: Debajo de la imagen cargada, se presenta una sección titulada "Similar Images", que muestra una selección de imágenes que la aplicación considera similares a la cargada, además de presentar los valores de índices y distancias respectivas.

Funcionalidades

1. **Carga de Imágenes**:

- Los usuarios pueden subir una imagen haciendo clic en el botón "UPLOAD PHOTO". Esto activa un diálogo de selección de archivos donde pueden elegir una imagen de su dispositivo.

2. **Visualización de la Imagen Cargada**:
   - Una vez cargada la imagen, se muestra en un formato grande en el centro de la página para que el usuario pueda verificar qué imagen está siendo analizada.
3. **Búsqueda y Visualización de Imágenes Similares**:
   - La aplicación busca imágenes similares cuando la imagen se ha cargado y procesado.
   - Las imágenes que se determinan como similares se muestran en miniatura debajo de la imagen principal. Estas miniaturas permiten al usuario ver rápidamente otras imágenes que comparten características visuales con la subida.

Posibles Mejoras

- **Filtros y Opciones de Búsqueda**: Añadir funcionalidades para ajustar los criterios de búsqueda de imágenes similares, como cambiar la importancia de ciertos atributos visuales.
- **Información Adicional**: Mostrar detalles sobre las imágenes similares, como su origen, derechos de autor, o descripciones.
- **Interacción Mejorada**: Permitir a los usuarios marcar si las sugerencias de imágenes similares fueron útiles o no para mejorar el algoritmo de búsqueda.

Anexos

Repositorio