# ESCUELA POLITÉCNICA NACIONAL

# FACULTAD DE INGENIERÍA DE SISTEMAS

# RECUPERACIÓN DE LA INFORMACIÓN

## Integrantes:


- RAQUEL ZUMBA
- RAQUEL ZUMBA


**DOCENTE:** IVAN CARRERA

**FECHA DE ENTREGA:** 13-08-2024

# 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 dividirá en varias fases, que se describen a continuación.

# 2 Fases del proyecto

## 2.1 Adquisición de datos

- Objetivo: Obtener y preparar el dataset Caltech101.

En la ilustración 1 lo que se realiza es la división del dataset para el entrenamiento tomamos el 80% y para la prueba el 20%, luego de eso guardamos la data en la ruta que se especifique.

```
[30] (train_dataset, test_dataset), dataset_info = tfds.load(
        # Nombre del dataset a cargar
        name='caltech101',
        # División del dataset: 80% para entrenamiento, 20% para prueba
        split=['train[:80%]', 'test[20%:]'],
        # Incluye información adicional del dataset
        with_info=True,
        as_supervised=True,
         # Directorio donde se guardará el dataset
         data_dir='/content/drive/My Drive/DataProyectoIIB/'
        #download=False
    )
```

*Ilustración 1 Descargar el dataset*

En la ilustración 2 se muestra el número de clases que contiene la data, como se puede observar tenemos 102 clases en la data.

```
[31] # Obtener el número de clases del conjunto de datos
     num_classes = dataset_info.features['label'].num_classes
     # Imprimir el número de clases
     print(f'Número de clases en el dataset: {num_classes}')

     Número de clases en el dataset: 102
```

*Ilustración 2 Numero de clases*

En la ilustración 3 se muestra el total de imágenes por cada clase, esto se hizo para ver si la data estaba balanceada.

```
[32] # Inicializar un diccionario para contar el número de imágenes por clase
     class_counts = {i: 0 for i in range(num_classes)}

     # Contar las imágenes por clase en el conjunto de entrenamiento
     for image, label in tfds.as_numpy(train_dataset):
         class_counts[label] += 1

     # Contar las imágenes por clase en el conjunto de prueba
     for image, label in tfds.as_numpy(test_dataset):
         class_counts[label] += 1

     # Obtener los nombres de las clases
     class_names = dataset_info.features['label'].int2str

     # Imprimir el número de imágenes por clase
     for class_id, count in class_counts.items():
         class_name = class_names(class_id)
         print(f'Clase: {class_name}, Número de imágenes: {count}')
```

```
Clase: accordion, Número de imágenes: 43
Clase: airplanes, Número de imágenes: 641
Clase: anchor, Número de imágenes: 33
Clase: ant, Número de imágenes: 37
Clase: background_google, Número de imágenes: 368
Clase: barrel, Número de imágenes: 38
Clase: bass, Número de imágenes: 47
```

*Ilustración 3 Total de imágenes por clase*

## 2.2 Preprocesamiento

- Objetivo: Preparar los datos para su análisis.

En la Ilustración 4, se lleva a cabo el preprocesamiento de imágenes, que consta de dos etapas principales. Primero, las imágenes se redimensionan a un tamaño de 224x224 píxeles, lo cual asegura que todas las imágenes tengan dimensiones uniformes y sean compatibles con el modelo. Luego, se realiza una normalización de los valores de los píxeles, transformando los valores enteros de rango de 0 a 1.

Preprocesar las imagenes del dataset

```
[44] # Función que toma una imagen y una etiqueta como entrada
     def preprocess_image(image, label):
       #Redimensiona cada imagen a un tamaño de 224x224 píxeles
         image = tf.image.resize(image, (224, 224))
         #Esta línea convierte los valores de los píxeles de la imagen de enteros de 0-255 a flotantes de 0-1
         image = tf.cast(image, tf.float32) / 255.0
         #Retorna la imagen y la etiqueta
         return image, label
```

*Ilustración 4 Preprocesar las imágenes*

En la ilustración 5 se aplica la función de preprocesamiento a cada imagen en el conjunto de datos de entrenamiento y prueba.

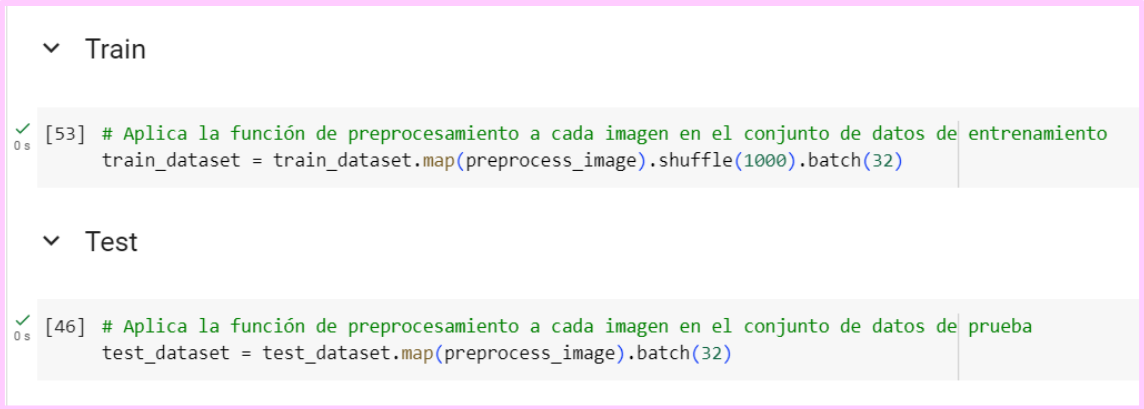


*Ilustración 5 Preprocesamiento del conjunto de datos*

## 2.3 Extracción de características

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

### 2.3.1 Cargar modelo Preentrenado

En la ilustración 6 se carga el modelo VGG16 preentrenado con pesos de ImageNet, excluyendo la capa de clasificación final para que solo se utilicen las capas convolucionales. Luego, crea un nuevo modelo que utiliza la entrada del VGG16 base y genera los mapas de características a partir de la última capa convolucional.

```
# Load the VGG16 model with pretrained weights from ImageNet, without the top classification layer
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

#Crea un nuevo modelo que genere los mapas de características.
model = Model(inputs=base_model.input, outputs=base_model.layers[-1].output)
```

*Ilustración 6 Cargar el modelo preentrenado*

### 2.3.2 Función para extraer características

La función `extract_features` toma un conjunto de datos como entrada y extrae características utilizando un modelo predefinido. Inicializa listas vacías para almacenar las características extraídas, las etiquetas y las imágenes originales. Luego, recorre cada lote de imágenes y etiquetas en el conjunto de datos, almacenando las imágenes en una lista, prediciendo los mapas de características

con el modelo para cada lote, y almacenando estas características en otra lista. Finalmente, convierte las etiquetas a formato numpy y las guarda en una lista separada.

```
# Función para extraer características del conjunto de datos
def extract_features(dataset):
    # Inicializa listas vacías para almacenar características, etiquetas e imágenes
    features = []  # Lista para almacenar los mapas de características extraídos de cada imagen
    labels = []    # Lista para almacenar las etiquetas correspondientes a cada imagen
    imgs = []    # Lista para almacenar las imágenes originales
    # Itera sobre cada par de imágenes y etiquetas en el conjunto de datos
    for images, lbls in dataset:
        imgs.append(images)  # Agrega las imágenes a la lista imgs
        # Utiliza el modelo para predecir las características de las imágenes
        feature_maps = model.predict(images)
        features.append(feature_maps)  # Agrega los mapas de características a la lista features
        labels.append(lbls.numpy())     # Convierte las etiquetas a un formato numpy y las agrega a la lista labels

    # Devuelve las listas de características, etiquetas e imágenes
    return features, labels, imgs
```

*Ilustración 7 Extraer características*

La ilustración extrae características para los conjuntos de datos de entrenamiento y prueba utilizando la función extract_features. Llama a esta función primero con el conjunto de datos de entrenamiento (train_dataset), almacenando los resultados en train_features, train_labels y train_img. Luego, hace lo mismo para el conjunto de datos de prueba (test_dataset), guardando los resultados en test_features, test_labels y test_img. Esto permite obtener las características y etiquetas para ambos conjuntos de datos.

```
# Extraer características para los conjuntos de datos de entrenamiento y prueba
train_features, train_labels,train_img = extract_features(train_dataset)
test_features, test_labels,test_img = extract_features(test_dataset)
```

*Ilustración 8 Extracción de características*

## 2.4 Indexación

El código aplana los mapas de características, etiquetas e imágenes para los conjuntos de entrenamiento y prueba. Primero, convierte los mapas de características del conjunto de entrenamiento en vectores unidimensionales y guarda los resultados en *train_features_flat*. Luego hace lo mismo con las etiquetas y las imágenes, guardando en *train_labels_flat* y *train_img_flat*.

```python
# Aplanar los mapas de características para el conjunto de entrenamiento
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.numpy().flatten() for batch in train_img for img in batch])

# Aplanar los mapas de características para el conjunto de prueba
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.numpy().flatten() for batch in test_img for img in batch])
```

*Ilustración 9 Aplanamiento de características para train y test*

### 2.4.1 Guardar las características, etiquetas e imágenes aplanadas para el conjunto de entrenamiento y prueba.

El código guarda las características, etiquetas e imágenes aplanadas de los conjuntos de entrenamiento y prueba en archivos `.npy`. Luego imprime un mensaje para confirmar que se han guardado correctamente.

```python
# Guardar las características, etiquetas e imágenes aplanadas para el conjunto de entrenamiento
np.save('train_features.npy', train_features_flat)
np.save('train_labels.npy', train_labels_flat)
np.save('train_imgs.npy', train_img_flat)

# Guardar las características, etiquetas e imágenes aplanadas para el conjunto de prueba
np.save('test_features.npy', test_features_flat)
np.save('test_labels.npy', test_labels_flat)
np.save('test_imgs.npy', test_img_flat)

print("Las características, etiquetas e imágenes aplanadas han sido guardadas correctamente.")
```

*Ilustración 10 Guarda las características*

### 2.4.2 Cargar las características, etiquetas e imágenes aplanadas para el conjunto de entrenamiento.

El código carga las características, etiquetas e imágenes aplanadas para los conjuntos de entrenamiento y prueba desde los archivos `.npy` guardados previamente. Luego almacena estos datos en las variables correspondientes para su uso posterior.

```python
import numpy as np

# Cargar las características, etiquetas e imágenes aplanadas para el conjunto de entrenamiento
train_features_flat = np.load('train_features.npy')
train_labels_flat = np.load('train_labels.npy')
train_imgs_flat = np.load('train_imgs.npy')

# Cargar las características, etiquetas e imágenes aplanadas para el conjunto de prueba
test_features_flat = np.load('test_features.npy')
test_labels_flat = np.load('test_labels.npy')
test_imgs_flat = np.load('test_imgs.npy')
```

*Ilustración 11 Cargar las características*

## 2.5 Motor de Búsqueda

El código entrena un modelo de vecinos más cercanos con 5 vecinos y el algoritmo `ball_tree` usando las características aplanadas del conjunto de entrenamiento. Esto permite realizar búsquedas de vecinos más cercanos en base a las características.

```python
nn_model = NearestNeighbors(n_neighbors=5, algorithm='ball_tree').fit(train_features_flat)
```

### 2.5.1 Obtener las características y la imagen de consulta

En la ilustración 12 se empieza por seleccionar la imagen con el índice 5 del conjunto de prueba y obtiene sus características y la imagen. Luego, redimensiona las características para la búsqueda. A continuación, utiliza un modelo de vecinos más cercanos para encontrar las 10 imágenes más similares en el conjunto de prueba, devolviendo las distancias y los índices de estos vecinos.

```python
# Seleccionar la imagen con índice 10
query_index = 5

# Obtener las características y la imagen de consulta
query_features = test_features_flat[query_index]
query_img = test_imgs_flat[query_index]

# Redimensionar las características de la imagen de consulta para la búsqueda
query_features = query_features.reshape(1, -1)

# Encontrar los vecinos más cercanos en el conjunto de prueba
from sklearn.neighbors import NearestNeighbors
nn_model = NearestNeighbors(n_neighbors=10, algorithm='ball_tree').fit(test_features_flat)
distances, indices = nn_model.kneighbors(query_features)
```

*Ilustración 12 Seleccionar el índice de la imagen*

### 2.5.2 Mostrar la imagen de consulta

En la ilustración 14 se muestra la imagen de consulta que en este caso pertenece a la clase flamenco.

```
# Mostrar la imagen de consulta
query_img_display = query_img.reshape((224, 224, 3))  # Ajusta el reshape según el tamaño original
plt.imshow(query_img_display)
plt.title("Imagen de Consulta")
plt.axis('off')
plt.show()
```


*Ilustración 13 Mostrar la imagen de consulta*

### 2.5.3 Resultados de los índices y las distancias similares

La imagen 14 muestra el índice de la imagen de consulta en el conjunto de prueba, seguido de los índices y las distancias de las 10 imágenes más similares encontradas.

```
# Imprimir resultados
print(f"Índice de la imagen de consulta en el conjunto de prueba: {query_index}")
print("Índices de las imágenes más similares:", indices)
print("Distancias de las imágenes más similares:", distances)
```

*Ilustración 14 Impresión de los resultados de los índices*

### 2.5.4 Mostrar las imágenes similares encontradas

En la ilustración 15 se define una función ***display_images*** que muestra la imagen de consulta junto con las imágenes más similares encontradas

```python
# Mostrar las imágenes similares encontradas
def display_images(query_img, indices, dataset_images):
    """
    Muestra la imagen de consulta y las imágenes más similares encontradas.
    """
    fig, axes = plt.subplots(1, len(indices[0]) + 1, figsize=(15, 5))
    axes[0].imshow(query_img)
    axes[0].set_title("Consulta")
    axes[0].axis('off')

    for ax, idx in zip(axes[1:], indices[0]):
        img = dataset_images[idx].reshape((224, 224, 3))  # Ajusta la forma si es necesario
        ax.imshow(img)
        ax.axis('off')

    plt.show()

# Mostrar la imagen de consulta y las imágenes similares encontradas
display_images(query_img_display, indices, test_imgs_flat)
```

*Ilustración 15 Mostrar las imágenes similares*

En la ilustración 16 se muestra las imágenes similares de acuerdo a la consulta que se realizó, en este caso se puso para que se muestren las tres imágenes más cercanas.


*Ilustración 16 Imágenes Similares encontradas*