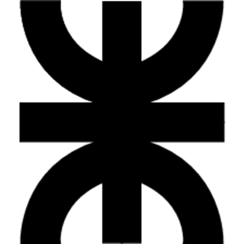
**UNIVERSIDAD TECNOLÓGICA NACIONAL FACULTAD REGIONAL CONCEPCIÓN DEL URUGUAY**



Gestión Avanzada de Datos

Trabajo Final

Docentes:

Ing. Pascal, Andrés

Ing. Planas, Adrián

Alumnos:

Aguirre, Lía Lucía

Balsanyaque, Lautaro

Chichi, Manuel

Richieri, Emanuel

Introducción

El presente trabajo tiene como propósito consolidar los conocimientos adquiridos a lo largo de la cursada de la asignatura, aplicando los conceptos, algoritmos, técnicas, etc. que se han presentado en la misma.

Para dicho fin se ha desarrollado una aplicación que permite cargar imágenes a la base de datos y realizar consultas por similitud respecto de las imágenes cargadas. En particular, en este caso, se ha cargado un conjunto de datos inicial de imágenes de perros de distintas razas, con el objetivo determinar los perros similares a una imagen ingresada como consulta.

A lo largo de este texto se describirán las distintas tecnologías y técnicas utilizadas para el desarrollo de la aplicación y de las funciones dentro de la base de datos. Además, se presentarán los resultados de pruebas realizadas con diversas imágenes de consulta.

Desarrollo

# Conjunto de datos

El conjunto de imágenes seleccionado para extraer características está conformado por imágenes de perros adultos, las cuales fueron extraídas de Google y de Unpash. Se seleccionaron 12 razas populares con un mínimo de 50 imágenes por raza, buscando diversidad de ángulos y pelaje para que los resultados sean más precisos. A su vez, se realizó una limpieza de los fondos de cada una de las imágenes. Para realizar esto se utilizó la página web remove.bg.

Las razas seleccionadas fueron las siguientes, conformando un total de 582 imágenes:

* Bulldog Frances
* Ovejero Aleman
* Bulldog Inglés
* Rottweiler
* Boxer
* Shar Pei
* Labrador retriever
* Golden Retriever
* Beagle
* Miniature Schnauzer
* Doberman
* Husky Siberiano

# **Base de datos**

Se utilizó el motor de base de datos PostgreSQL, sobre el cual se creó una base de datos en la que se crearon 4 tablas principales para realizar el trabajo:

* ***Tabla de imágenes***: contiene el vector de características de cada imagen con su nombre y el nodo al que pertenecen.
* ***Tabla de pivotes***: contiene vectores de características de ciertas imágenes seleccionadas junto con el nivel al que pertenecen.
* ***Tabla de intervalos***: contiene los dos intervalos requeridos por el índice y un arreglo con los intervalos padres.
* ***Tabla de índice:*** contiene a qué intervalo pertenece la imagen y la distancia que existe con el pivote del nivel.

# Extracción del vector de características

Para extraer el vector de características de una imagen, se requiere previamente que las imágenes se encuentren limpias (sin fondo) y redimensionadas. En este caso se ha conformado un script en ***Python*** que permite realizar la limpieza, redimensión y extracción del vector de características de una imagen para consultas. En el caso de la carga masiva de imágenes y pivotes en la base de datos no se realiza la limpieza ya que sería muy costoso. Para este último caso se recomienda limpiar las imágenes anteriormente.

Para la limpieza de la imagen de consulta se utiliza una **API** brindada por ***remove.bg***. En esta se sube la imagen y luego el servicio se encarga de extraer los bordes de la misma dejando solo el objeto. Para realizar esto, según la documentacion oficial de *remove.bg*, se utilizan redes neuronales. Al realizarlo de manera automática, la limpieza puede tener errores.

Una vez obtenida la imagen limpia, se la redimensiona (224x224 px) y se procede a extraer el vector de características. Para este caso, se ha utilizado la librería ***img2vec*** que se encarga de transformar una imagen a un vector de números reales el cual cuenta con un tamaño de 512 valores. Posteriormente, este es insertado a la base de datos. La librería *img2vec* utiliza ***Pytorch*** como principal herramienta para formar este vector. Dentro de *Pytorch* existen distintos modelos de redes convolucionales ya entrenadas con una basta cantidad de imágenes que simplifican las tareas al no necesitar entrenar la red. De esta forma, *img2vec nos permite* utilizar ***resnet-18*** como ***red convolucional***. Esta red convolucional está compuesta por 18 capas convolucionales y se caracteriza por permitir realizar saltos entre las neuronas de las distintas capas.

Como hemos mencionado anteriormente, una vez obtenido el vector de características este se inserta en la tabla de imágenes para permitir realizar las consultas.

# **Función de distancia**

Para la función de distancia se realizó una función en PostgreSQL que toma como parámetros dos vectores de características y arroja como resultado la distancia euclidiana entre los mismos.

**Dimensionalidad intrínseca**

Una vez conformado el conjunto de datos y su función de distancia, se calculó la dimensionalidad intrínseca de los elementos. Para esto se calculó la media y la desviación estándar de la distancia entre todos los vectores de características de las imágenes. Esto arrojó que la media de la distancia es de *22.4538* y la desviación estándar de *3,1777*.

De esta forma, tras aplicar la fórmula provista por la cátedra (), la ***dimensión intrínseca da el valor 24,9687***. Puesto que es mayor que 20, este no es un valor que garantice un buen funcionamiento del índice.

# **Índice**

Para reducir la cantidad de evaluaciones de función de distancia que se realiza en cada consulta, se utilizó el índice FHQT+. El mismo fue desarrollado también en PostgreSQL y se ejecuta a través de un trigger al momento de almacenar o actualizar cada imagen en la base de datos.

Este índice es una variante para valores continuos de distancia del FHQT, y funciona a partir de un árbol balanceado, en el cual los nodos interiores están conformados por intervalos que representan la distancia al pivote de su nivel. Cada nodo tiene dos intervalos: uno que representa el valor mínimo y el valor máximo posible de distancia al pivote que pueden contener los elementos de la rama, mientras que se conforma con los valores mínimo y máximo reales de los elementos que contiene la rama.

De esta manera, cuando se realiza una consulta se va calculando la distancia del elemento consultado contra cada pivote, y se va ingresando únicamente por aquellas ramas cuyos intervalos contengan a esa distancia más/menos el rango de la consulta, hasta llegar finalmente a los intervalos de los nodos hoja, que contendrán los candidatos a formar parte del resultado de la consulta.

Por último, se evalúa la función de distancia contra cada candidato y si la misma es menor que el rango de la consulta, el elemento se retorna como resultado.

## Selección de pivotes e intervalos

En base a la cantidad de elementos que se agregaron inicialmente a la base de datos (aproximadamente 600), se seleccionaron 7 pivotes, cada uno de los cuales se corresponde con la imagen de un perro de una raza distinta dentro de las más populares (Ovejero Alemán, Boxer, Bulldog Francés, Golden Retriever, Beagle, Husky Siberiano, Rottweiler).

Se eligió 3 como cantidad de intervalos por cada pivote, también considerando la cantidad de elementos de la base de datos.

Para determinar los intervalos se tomó una muestra aleatoria de 500 elementos por cada nivel, y utilizando la función NTILE de PostgreSQL -la cual divide la muestra en N intervalos (en este caso N=3) dejando la misma cantidad de elementos en cada uno de ellos- se obtuvieron los rangos mínimo y máximo de cada intervalo. Con esos intervalos, se completa el nivel del árbol correspondiente, asignando los mismos como hijos de cada intervalo del nivel anterior.

# **API**

Con el lenguaje de programación ***Python*** y la librería ***Flask*** se ha creado una API con los siguientes end-points.

* ***api/query***: se realiza una petición POST en formato JSON con una imagen en base64 y el rango de consulta como parámetros. Devuelve en formato JSON, un listado de imágenes base64 que satisfacen el rango, así como también la distancia de cada una de ellas a la imagen de consulta.
* ***api/loadImages***: se realiza una petición POST en formato JSON con un arreglo de imágenes en base64, las cuales son agregadas a la base de datos -disparando el trigger de actualización del índice-. Devuelve en formato JSON, un mensaje de éxito si no hubo errores.

# **Aplicación web**

Para la interfaz de usuario se desarrolló una aplicación web utilizando ***Ionic*** y ***Angular***. La misma posee dos funcionalidades principales: realizar una consulta y cargar imágenes a la base de datos. En ambos casos, se hace uso de la API desarrollada en Python.

Pruebas

## Precisión

Para poner a prueba la certeza de la herramienta se realizaron pruebas con un total de 40 imágenes, 5 imágenes por cada raza de perros, las cuales se detallan en la siguiente tabla. Se utilizó un rango igual a 1 y con una cantidad de máxima de imágenes igual a 5. La prueba se considera exitosa cuando la primera imagen de la búsqueda corresponde con la raza consultada.

|  |  |  |  |
| --- | --- | --- | --- |
| **Raza de perro de búsqueda** | **Aciertos** | **Fallos** | **Tasa de éxito (%)** |
| Labrador | 2 | 3 | 40 |
| Rottweiler | 5 | 5 | 100 |
| Golden Retriver | 5 | 0 | 100 |
| Husky Siberiano | 4 | 1 | 80 |
| Beagle | 5 | 0 | 100 |
| Bulldog Frances | 4 | 1 | 80 |
| Boxer | 4 | 1 | 80 |
| Pastor Alemán | 5 | 5 | 100 |
|  |  | **Promedio** | **85** |

En esta prueba con una pequeña cantidad de datos, se obtuvo que el 85% de las imágenes coincidía con las razas buscadas. Para evaluar con mayor certeza la precisión de la herramienta podría considerarse aumentar la cantidad de búsquedas y realizar el mismo análisis.

Se analizó con detalle el caso de la raza labrador y se constató que esta raza es altamente similar con los golden retriever, lo que explica su baja tasa de éxito.

## Robustez

Para comprobar la robustez de la herramienta se ha tomado aleatoriamente la imagen de un perro y a la misma se la han realizado diversas modificaciones para obtener un conjunto de imágenes de prueba. Para esta evaluación, consideramos el caso de prueba como exitoso cuando la imagen del perro aparece en las 5 primeras imágenes no modificadas (es decir 5 más cercanas).

Los resultados arrojaron que al realizar cambios de colores no muy bruscos, como por ejemplo saturar un color, la herramienta sigue detectando correctamente al perro y su raza. Si bien la distancia aumenta, no lo hace lo suficiente como para que el caso de prueba fracase. Lo mismo sucede al convertir a escala de grises los colores de la imagen. Al difuminar la imagen se obtienen resultados similares.

Por el contrario, en los casos donde se ha modificado bruscamente la paleta de colores, pixelado la imagen o con rotaciones abruptas, los casos de prueba fracasaron. A continuación se adjuntan algunos ejemplos de las modificaciones. La imagen original es la primera de izquierda a derecha.



Conclusiones

La utilización de índices mejora notablemente la cantidad de comparaciones que se deben realizar. Para una búsqueda, en vez de realizar 582 comparaciones (una por cada elemento de la base de datos), aplicando índices se ha reducido este número a 205 comparaciones en promedio. Estos valores podrían mejorarse aún más al utilizar una mayor cantidad de subdivisiones.

Por otra parte, para hacer más efectivo el índice podría considerarse reducir la dimensionalidad intrínseca. Una manera de realizar esto es quitando aquellos valores que no son lo suficientemente discriminatorios del vector de características. Se requeriría realizar un estudio exhaustivo en el cual se identifiquen dichos valores.

Agregar histogramas de colores al vector de características fue descartado ya que se podía observar que la red neuronal tomaba en cuenta estos factores. Además, considerando el valor de la dimensionalidad intrínseca con la cual contaba el conjunto de datos, agregar más valores al vector simplemente incrementaría la dimensionalidad y empeoraría el funcionamiento del índice.

Para mejorar la robustez podría entrenarse la red neuronal e introducir imágenes rotadas al conjunto de datos. Esto permitiría detectar una mayor cantidad de casos.

Finalmente, las pruebas de precisión otorgaron resultados positivos teniendo una efectividad de aproximadamente el 85%. De esta manera, se considera la herramienta es funcional y que cumple con los objetivos planteados.

Bibliografía

* Safka, C. (2017, 3 noviembre). Extract a feature vector for any image with PyTorch. Recuperado de <https://www.becominghuman.ai/extract-a-feature-vector-for-any-image-with-pytorch-9717561d1d4c>
* Rosebrock, A. (2020, 13 febrero). How-To: 3 Ways to Compare Histograms using OpenCV and Python - PyImageSearch. Recuperado de <https://www.pyimagesearch.com/2014/07/14/3-ways-compare-histograms-using-opencv-python/>
* Pascal A. et al. (2013, octubre). Una Extensión del FHQT Temporal para Distancias Continuas. Recuperado de <http://sedici.unlp.edu.ar/handle/10915/31294>.