Content-based image retrieval

ARISTOTLE UNIVERSITY OF THESSALONIKI

Advanced Topics in Databases
MSc Data and Web Sciences

Zisis Flokas
flokaszisis@csd.auth.gr
AEM: 73

Spring 2022

Introduction

Content-based image retrieval

Content-based image retrieval is a special case of Multimedia Information Retrieval (MIR) which belongs to the broader family of Information Retrieval (IR) systems. Other than images, subjects of MIR systems are audio and video.

In Content-based image retrieval our goal is given an input query image to find similar images based on their content. In order to perform this kind of similarity search we need to represent images in ways that make sense for the computer and we can perform some kind of calculation to measure similarity. In practice this is achieved by generating vectors often called descriptor vectors based on different techniques that encode characteristics of an image such as color, texture and shape in vector representations. Using these vectors we can calculate dissimilarity between two images using distance metrics like Euclidean distance. Having calculated the distance between the descriptor of a query image and those of a pool of images we can retrieve the k nearest neighbors (K-NN queries).

Calculating distance metrics between a large set of objects to rank can be proven really difficult and many methods such as approximation algorithms for distance calculation and improved indexing techniques have been proposed to address these issues. In the context of this application we do not take into account this kind of requirement as it is more of a proof of concept working over a fairly small dataset.

Application Overview

To implement Content-based image retrieval engine a Python application was created which provides a web UI for the user to upload his/her own images, select distance metric and desired number (K) of items to retrieve in order to perform the image retrieval. Other than that a minimal Python CLI is provided in order to initialize the table that hosts the metadata of image dataset along with commands to sample images and ingest them in the database as well as a command to delete all records stored. The implementation requires having access to a PostgreSQL database with a user having create, insert and delete privileges over the public schema (or just the pet_images table) of the configured database.

The application is organized in different Python modules which will be described in more detail in the section Content-based image retrieval application.

Dataset

The primary ingredient of this project is the dataset of images. Oxford-IIIT Pet Dataset open dataset was selected as it is a well known dataset with roughly 7000 images of cats and dogs. It is important that those images are already annotated regarding pet family and pet breed in the following format.

- pet family: Cat images have a filename which starts with a capital letter while for dogs filenames start with a lowercase letter.
- pet breed: First part of the filename until "_" character provides the pet breed in snake case format

Those annotations help us evaluate the performance of the search experiments we make in the following sections.

Dataset configuration instructions

In order to use this application the dataset must be downloaded and placed in a specific directory of the project. Clicking the following link will fire the downloading of a GZ file named **images.tar.gz** containing a compressed folder named **images** with all pet images. This GZ file must be decompressed in order to move the **images** folder it contains under the **data** folder of the project. The structure should be the following:

- root directory of project > data > images

The data/images folder is excluded from git versioning as it is added in the .gitignore of the project.

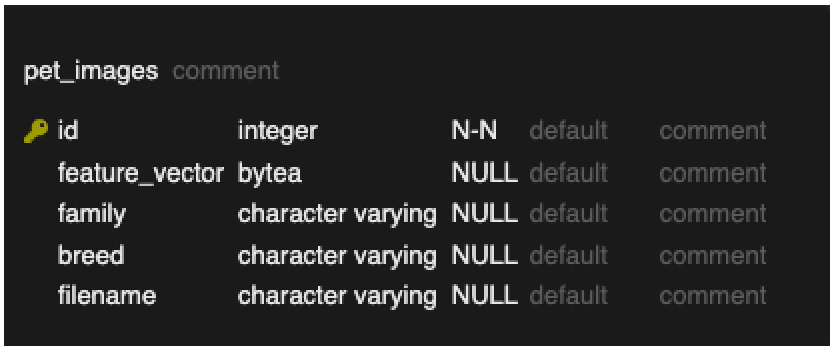
Dataset Statistics

In the dataset there are 4978 images of dogs and 2371 of cats. Each breed regardless of the family is represented with roughly 200 images. In the application only a sample of these photos is taken in a random way with no duplicates. We can sample images randomly and have a fairly evenly distributed set of breeds as the original dataset is balanced regarding this aspect. During sampling of images we pick the same number of images both for cat and dog families.

Database

For the backend storage of the application, PostgreSQL database was chosen as it is one of the most widely adopted relational databases. The database is used to create a single table named **pet_images** which will host the metadata of the dataset that will be used for the image retrieval.

The schema of this table is the following:



pet_images comment

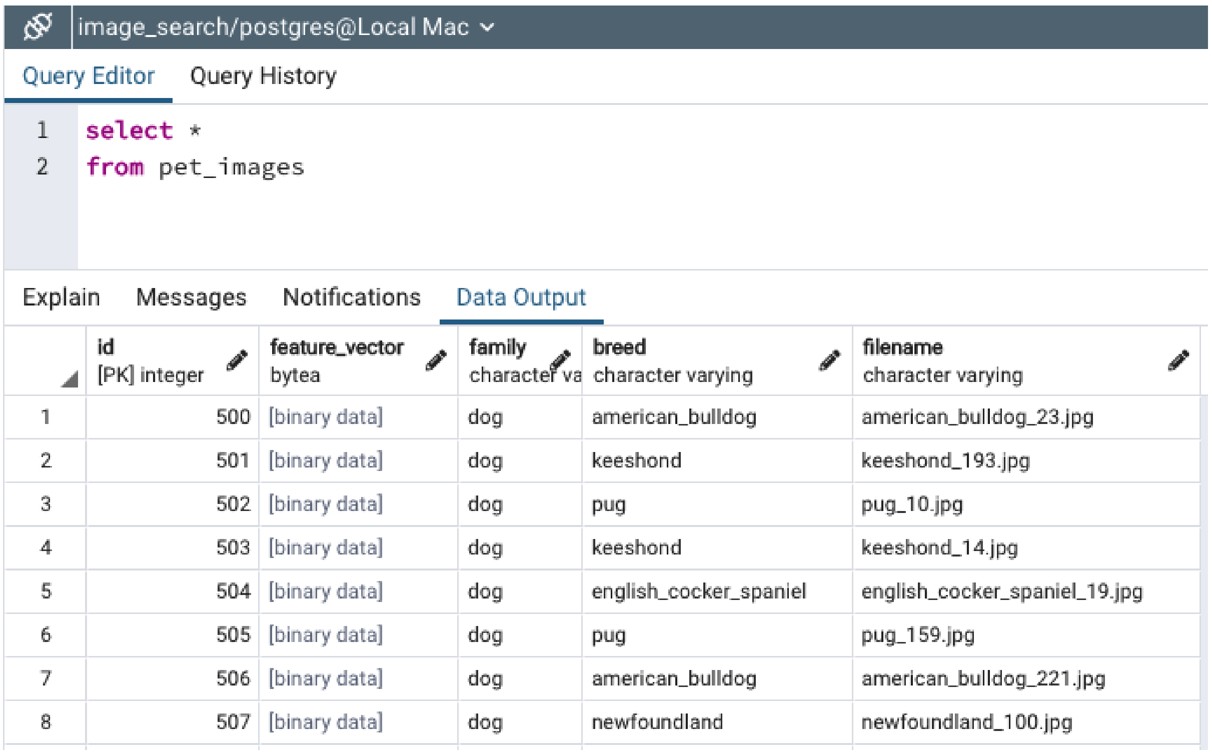
id	integer	N-N	default	comment
feature_vector	bytea	NULL	default	comment
family	character varying	NULL	default	comment
breed	character varying	NULL	default	comment
filename	character varying	NULL	default	comment

Columns description:

- id: A unique auto incrementing identifier

- feature_vector: Is a binary encoded representation of a Python numpy array created by the application (feature_generator python module) when ingesting rows to serve as the descriptor vector
- family: The family of the pet which can be either cat or dog
- breed: The breed of the pet in snake case format
- filename: Filename of original image

Performing a select query over all columns and rows in the tables pet_images



```
select *
from pet_images
```

	id [PK] integer	feature_vector bytea	family character va	breed character varying	filename character varying
1	500	[binary data]	dog	american_bulldog	american_bulldog_23.jpg
2	501	[binary data]	dog	keeshond	keeshond_193.jpg
3	502	[binary data]	dog	pug	pug_10.jpg
4	503	[binary data]	dog	keeshond	keeshond_14.jpg
5	504	[binary data]	dog	english_cocker_spaniel	english_cocker_spaniel_19.jpg
6	505	[binary data]	dog	pug	pug_159.jpg
7	506	[binary data]	dog	american_bulldog	american_bulldog_221.jpg
8	507	[binary data]	dog	newfoundland	newfoundland_100.jpg

Configuring connection

In order to configure connection of PostgreSQL database to the image retrieval Python application it is required to create a .env file in the root directory of the project. In that file the application expects to read five key value pairs of parameters required to set up connection. Those key value pairs are:

- DB_HOST: The host address of the database
- DB_PORT: The port number of the database on host
- DB_NAME: The name of the database that table will be created
- DB_USER: The use that the python application will use to perform actions
- DB_SECRET: The password of DB_USER

The final .env file may look like this:

```
DB_HOST=localhost
DB_PORT=5432
DB_NAME=image_search
DB_USER=postgres
```

```
DB_SECRET=postgres
```

Content-based image retrieval application

The Content-based image retrieval system was implemented entirely using Python. The application connects to the PostgreSQL database to initially create and afterwards fetch the table of pet images dataset to perform search. The Github repository of the project can be found in this link.

Application modules outline:

- app.py: Implements the web UI using streamlit Python library and the application logic making use of the modules described next
- db.py: Is the database handler for the connection with the PostgreSQL using SQLAlchemy package, it also provides a minimal CLI with 3 basic commands to interact with the database outside of the web UI. This module is also used to retrieve pet_images table in a form of Pandas dataframe
- feature_generator.py: Implements the PetImage class which is used to extract metadata for pet images such as descriptor vectors, pet breed and family etc.
- image_retrieval.py: Implements the QueryImage class which is responsible to generate metadata for and represent input query images provided by the user. Also in this module **retrieve_k_most_similar_images** function is implemented to perform retrieval logic for the images using different metrics over the Pandas dataframe created from db.py module
- image_sampling.py: Provides a function to randomly extract samples of images from the original dataset
- experiments.py: A module to automatically run three different image search experiments using all different metrics and k options. This module produces collages of result query images which are stored in data/experiment_results folder

Feature extraction for Content-based image retrieval

The feature vectors of the pet images were created using the pretrained convolutional neural network model VGG16 from Tensorflow implementation. VGG16 is a model trained on a large dataset of images in order to be used for Computer Vision applications such as image classification.
This model will be automatically downloaded as long as the relevant python packages are installed successfully when the user runs ingest action to populate the database for the first time.

Requirements

In order to use the application you need first to set up an appropriate Python virtual environment and install all the required packages.

Execute the following commands to create and activate a Python virtual environment using conda

```
conda create --name image_search_env python=3.8
conda activate image_search_env
```

Then install the package dependencies as listed in the requirements.txt file of the project

```
pip install -r base_requirements.txt
```

File base_requirements.txt contains all packages you need to install for this project except tensorflow related ones. This is because this project was developed in a Apple M1 machine and requires a different process to install tensorflow.

For Apple M1 do the following:

```
conda install -c apple tensorflow-deps
pip install tensorflow-macos
pip install tensorflow-metal
```

In other systems the following command should be enough:

```
pip install tensorflow
```

Database Handler CLI

In this project there is a Python module named db.py which as mentioned in the previous section serves as a handler for the PostgreSQL database. In order to interact with the database to create, ingest records for or delete records from the pet_images table a simple CLI is provided. Of course a prerequisite before using the CLI is to have configured the .env file in the Configuration connection section of Database chapter.

CLI commands

Execute the following command to create pet_images table that will host links to pet image files, metadata and feature vectors:

```
python db.py --action=create
```

Execute the following command to ingest a sample of roughly 250 cat and 250 dog image records in the pet_images table of PostgreSQL database. Images are randomly selected using take_image_files_sample from image_sampling.py module

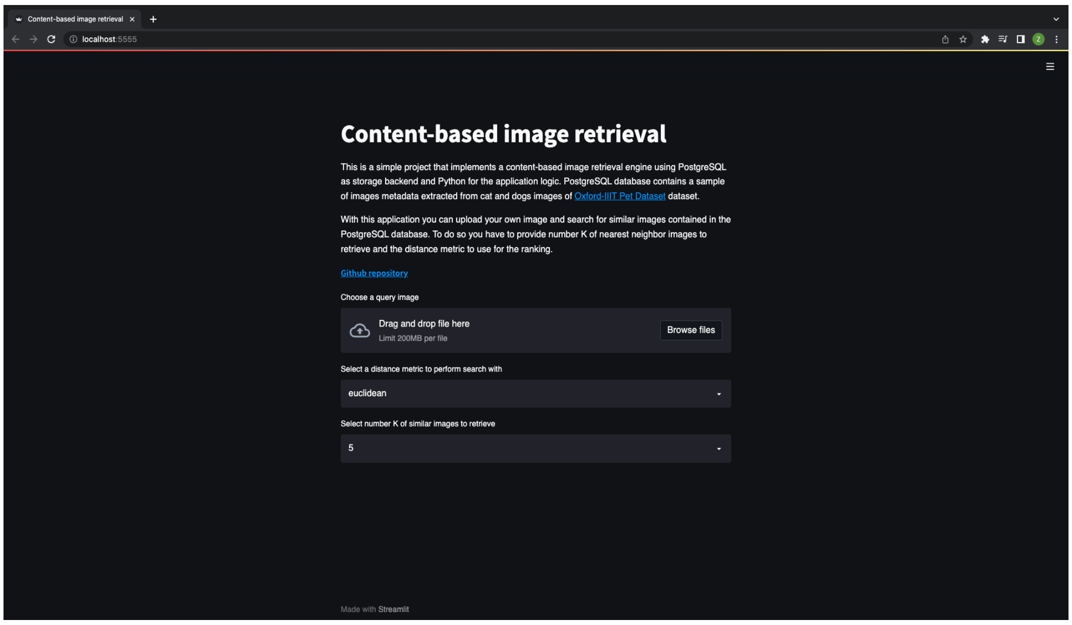
```
python db.py --action=ingest
```

Execute the following command to delete any image records of pet_images table

```
python db.py --action=delete
```

Image retrieval application UI

A Python web application was developed using the streamlit package in order to expose the method of image retrieval in a more user-friendly way. The web application has minimal functionality providing a description of the application, a placeholder to upload images and two dropdown menus one providing the different distance metrics (euclidean, cityblock, minkowski, chebyshev, cosine, canberra, jaccard) to use for ranking and one with the different available k values (5, 10, 20) for the K-NN query.



Content-based image retrieval

This is a simple project that implements a content-based image retrieval engine using PostgreSQL as storage backend and Python for the application logic. PostgreSQL database contains a sample of images metadata extracted from cat and dogs images of Oxford-IIIT Pet Dataset dataset.

With this application you can upload your own image and search for similar images contained in the PostgreSQL database. To do so you have to provide number K of nearest neighbor images to retrieve and the distance metric to use for the ranking.

Github repository

Choose a query image

Drag and drop file here
Limit 200MB per file
Browse files

Select a distance metric to perform search with

euclidean

Select number K of similar images to retrieve

5

Instructions to use the application

First you will need to create and activate a Python virtual env as described in section Requirements.

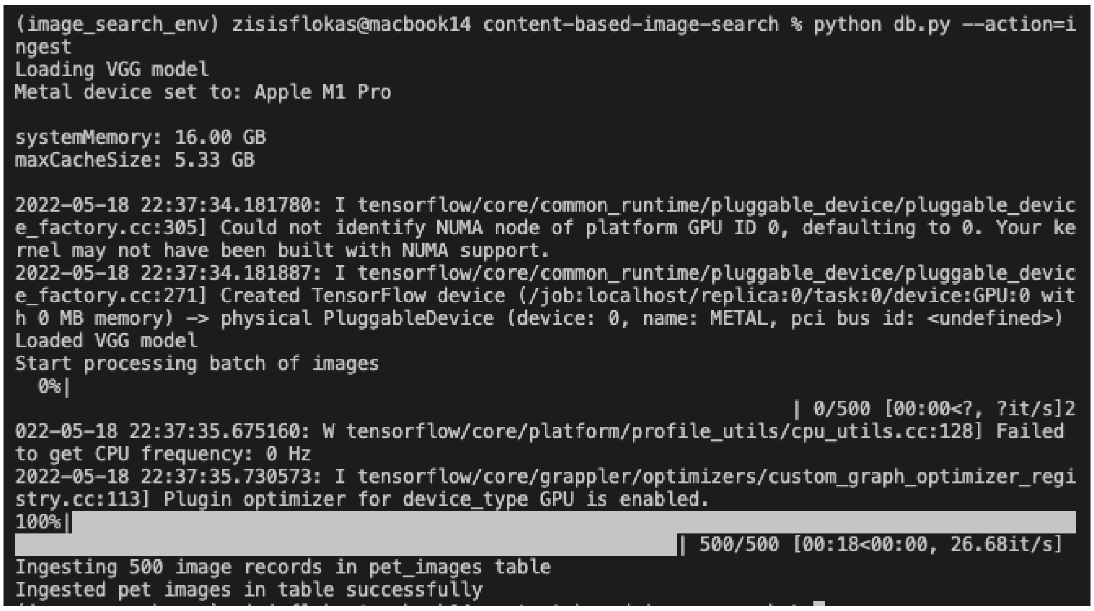
Second step is to create and populate the pet_images table in the PostgreSQL database. Execute the following command to create table in database

```
python db.py --action=create
```

Then execute the following command to ingest images metadata in the newly created table. There is a progress bar for the image processing phase which concerns feature vectors and metadata extraction. Some images may fail to be processed in this case appropriate messages will show up and those images will be excluded from the sample.

```
python db.py --action=ingest
```

The output of this command should look like this



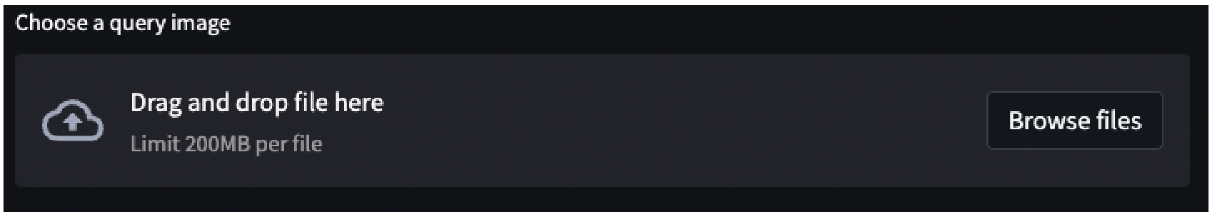
```
(image_search_env) zisisflokas@macbook14 content-based-image-search % python db.py --action=i
ngest
Loading VGG model
Metal device set to: Apple M1 Pro

systemMemory: 16.00 GB
maxCacheSize: 5.33 GB

2022-05-18 22:37:34.181780: I tensorflow/core/common_runtime/pluggable_device/pluggable_devic
e_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your ke
rnel may not have been built with NUMA support.
2022-05-18 22:37:34.181887: I tensorflow/core/common_runtime/pluggable_device/pluggable_devic
e_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 wit
h 0 MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
Loaded VGG model
Start processing batch of images
  0%|
                                                                   | 0/500 [00:00<?, ?it/s]2
022-05-18 22:37:35.675160: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed
to get CPU frequency: 0 Hz
2022-05-18 22:37:35.730573: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_regi
stry.cc:113] Plugin optimizer for device_type GPU is enabled.
100%|
                                                     | 500/500 [00:18<00:00, 26.68it/s]
Ingesting 500 image records in pet_images table
Ingested pet images in table successfully
```

As the database is ready to use you can start up the Python web application. In order to do that activate the Python virtual environment as described in section Requirements and execute the following command

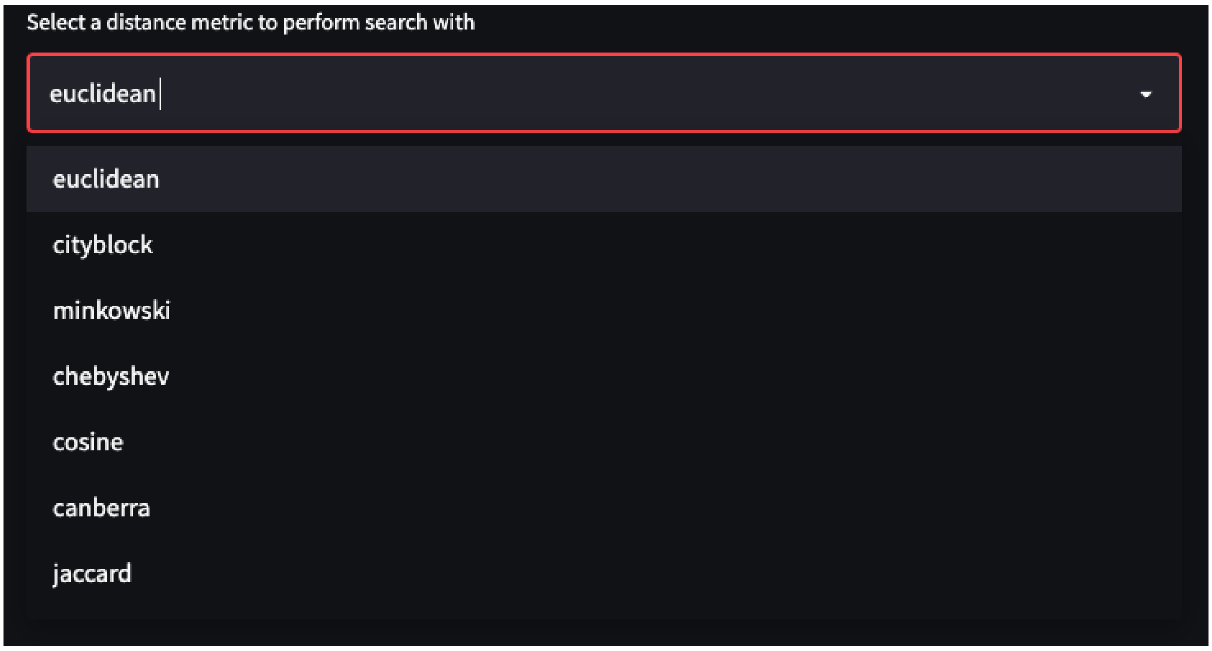
```
streamlit run app.py --server.port=5555
```

You can choose a different port to start the application if 5555 is taken. When the command is executed the web application will open up automatically in your web browser otherwise you can use localhost:5555 to access it.

Once the application is running in the browser you can perform the image retrieval by giving the appropriate input to the application. Choose an image to upload using the file upload placeholder



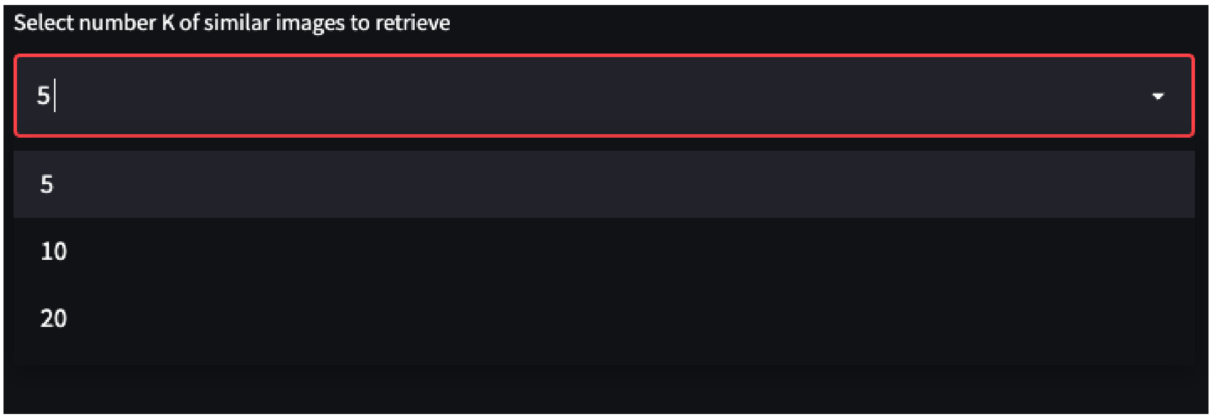
Then select a distance metric to use for ranking


Select a distance metric to perform search with

euclidean

- euclidean
- cityblock
- minkowski
- chebyshev
- cosine
- canberra
- jaccard

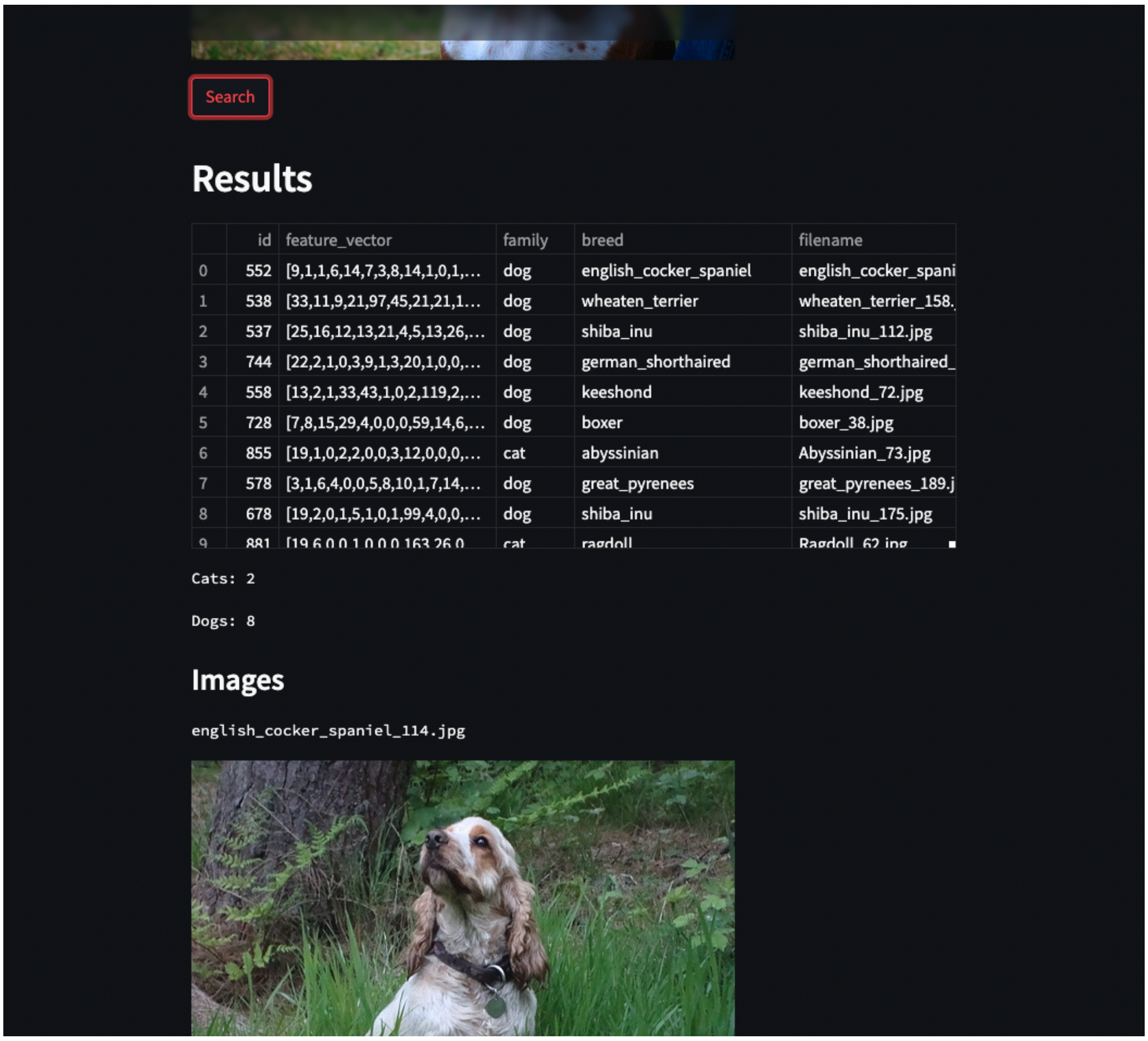
Select the number of results to retrieve


Select number K of similar images to retrieve

5

- 5
- 10
- 20

Finally press the search button to perform image retrieval. A results section is created with a table of the results ranked from most similar to least similar image and two counters for the number of cat and dog images retrieved. There is also a subsection that has all the related images for the user to spectate.



Results

	id	feature_vector	family	breed	filename
0	552	[9,1,1,6,14,7,3,8,14,1,0,1,…	dog	english_cocker_spaniel	english_cocker_spani
1	538	[33,11,9,21,97,45,21,21,1…	dog	wheaten_terrier	wheaten_terrier_158.
2	537	[25,16,12,13,21,4,5,13,26,…	dog	shiba_inu	shiba_inu_112.jpg
3	744	[22,2,1,0,3,9,1,3,20,1,0,0,…	dog	german_shorthaired	german_shorthaired_
4	558	[13,2,1,33,43,1,0,2,119,2,…	dog	keeshond	keeshond_72.jpg
5	728	[7,8,15,29,4,0,0,0,59,14,6,…	dog	boxer	boxer_38.jpg
6	855	[19,1,0,2,2,0,0,3,12,0,0,0,…	cat	abyssinian	Abyssinian_73.jpg
7	578	[3,1,6,4,0,0,5,8,10,1,7,14,…	dog	great_pyrenees	great_pyrenees_189.j
8	678	[19,2,0,1,5,1,0,1,99,4,0,0,…	dog	shiba_inu	shiba_inu_175.jpg
9	881	[19,6,0,0,1,0,0,0,163,26,0	cat	ragdoll	Ragdoll_62.jpg

```
Cats: 2

Dogs: 8
```

Images

```
english_cocker_spaniel_114.jpg
```

Experiments

To facilitate experiments a dedicated python module named experiments.py was created. This module when is executed performs image search using three different images and scores the results for all the different distance metrics implemented and for all the different options of k. Furthermore for each combination of metric and k option creates a collage of the query image results returned. Those image collages can be found in the data/experiments_results folder of the project.

The images used for the experiments as queries are those in folder data/sample_query_images. There are two images coming from the original dataset, dog image basset_hound_113.jpg and cat image Bengal_29.jpg. Those images may or may not take part in the database of images created in the PostgreSQL database because sampling is made randomly. There is also an image that doesn't take part in the original dataset and it is a dog of breed pug which is a breed that exists in our dataset the image is named out_of_dataset_dog.jpeg.

The distance metrics implemented for image retrieval come from the scipy python package and are the following:

- euclidean

- cityblock
- minkowski
- chebyshev
- cosine
- canberra
- jaccard

Performance metrics

For each experiment we calculate pet family and pet breed precision. We expect pet family precision to be higher as it is easier to make distinction between dogs and cats than specific breeds e.g. basset hounds and beagles. For the query image pet family and pet breed is provided as a known value, for the pool of possible results it is generated when pet images are ingested in the database.

Results

Next we provide the results for the three experiments and we mention the metric with the best performance.

Overall no metric was a clear winner but the most consistent ones with high scores were euclidean, cityblock, chebyshev and cosine. Jaccard performed the worst in all cases which makes sense as it tries to find exact value matches between two vectors.

More experiment results can be found in the Github repo of the project.

Case 1 basset hound dog from original dataset

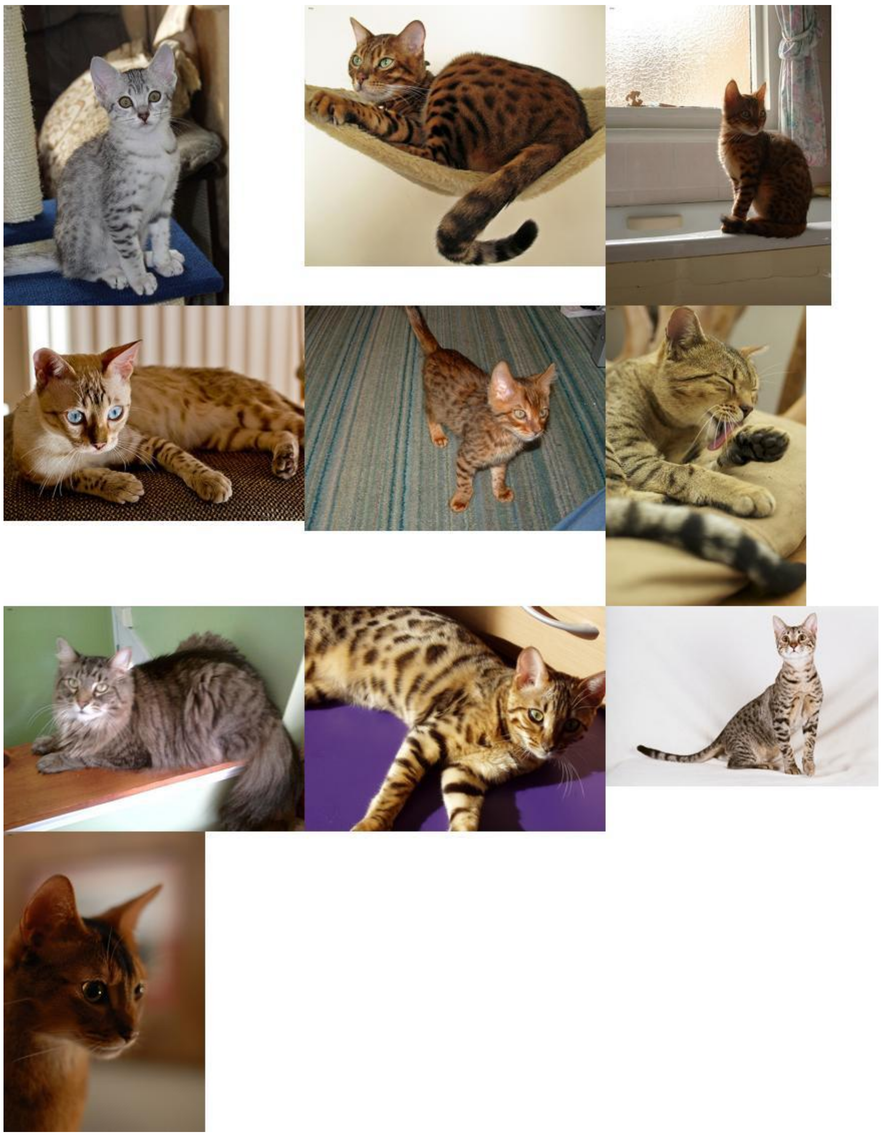
Query image

Results collage for k=10 and euclidean metric achieving precision 1 both for breed and family

Case 2 bengal cat from original dataset

Query image

Results collage for k=10 and chebyshev metric which was the best metric achieving precision 1 for family and 0.8 for breed



Case 3 pug dog out of the original dataset

Query image

Results collage for k=10 and cityblock metric which was the best metric achieving precision 1 for family and 0.7 for breed