Rishabh Jain
2019CSB1286

Documentation- Assignment 2

Brief description:

The program is an image search API implemented using flask and face_recognition API. Users can upload ZIP files or images into the database, and later give a sample image and find its matches in the database.

First, using the face_recognition API, an image encoding is saved into the database to optimize storage space and search. The user can then query an image. An encoding for this image will be generated, which will then be compared with the database encodings with a given confidence value. The top-K matches will be returned to the user.

Directory structure:

```
├── api
│   ├── __init__.py
│   ├── face_processing.py
│   ├── file_manager.py
│   ├── heap.py
│   └── models.py
├── main.py
├── requirements.txt
├── static
├── templates
│   ├── get_face_info.html
│   ├── home.html
│   ├── search_faces.html
│   ├── set_face_info.html
│   ├── show_results.html
│   ├── upload_image.html
│   └── upload_zip.html
├── testing_articles
├── testzip.zip
└── unit_test.py
```

Description of routes:

1. ‘/’(home): This is the route to access html template to use the API interactively.

2. ‘/add_face’: Route to upload an image. User needs to upload an image and a name of the person through a POST request.
   Form format:
   {‘file’:<FileObject>, ‘name’: <PersonNameAsText>}

   If the face-recognition library does not detect a face, the image is simply ignored.

3. ‘/add_faces_in_bulk’: User needs to upload a ZIP file through a post request.
   Form format:
   {‘file’:<ZIPfile>}
   ZIP file structure has to be of EXACTLY this form:

   ```
   <NameOfZipFile>.zip/ (this is the name of the zip file)

                        /<Person_1_Name>/image_1_of_person1
                                        /image_2_of_person1

                        /<Person_2_Name>/image_1_of_person2
                                        /image_2_of_person2
   ```

   That is, the zip file must be of utmost 2 levels deep, and every image must be inside a subfolder with a name of the person whose images are inside the folder. Refer to testzip.zip file for reference.

   This zip file is unzipped, stored in a unique folder in /static folder, and then the files loaded onto the database.
   The zip file must not contain any other files other than images(preferably).

4. ‘/search_faces’: Route to send a POST request to search the database using an image.
   Form format:
   {
   ‘image’:<ImageObject>,
   ‘k’:<K parameter as int’,
   ‘confidence’:< A number between 0 and 1>
   }

Returns results in a JSON object.
Note that if less than K images are found in the database for the given confidence level, then only those many images are returned. The user should thus lower the confidence level to get more images.

5. ‘/get_face_info’: Route to get metadata of the image with a id ‘id’
   Send a POST request as a form.
   Format:
   {‘id’:<image ID in database>}
   Returns image metadata(if added) and the image path. If metadata other than name is not set, null is returned corresponding to that field.

6. ‘/set_face_info’: Route to set metadata of the image. Note that name can only be set while uploading the image to the database. Other metadata, which are date, location, and version of the image can only be added separately here through this route.
   Metadata can only be set once.
   Send POST request as follows:
   {
   ‘id’: <ID of image>,
   ‘date’:<Date as String>
   ‘version’:<Version as String>,
   ‘location’:<Location as String>
   }

SOLID principles used:

1. Single responsibility principle: Each class/module is responsible for a single functionality.
   - main.py: Flask router manager.
   - face_processing.py: Manages face processing
   - file_manager.py: Manages loading files(images and zip files) and saving.
   - heap.py: Manages top-K selection.
   - models.py: Manages Database tables.
2. Liskov Substitution principle: Database model classes are inherited from db.Model. Objects of this class can be directly sent as arguments to the db.session.add() function without specifying the class and are added to the appropriate table in the database. This thus follows LSP.

3. Dependency inversion: Classes such as FileManager and FaceProcessing abstract away minute details of file and image handling.