Student: Elshan Naghizade

Project: Python Library To Transform Image Datasets in Query-able SQL Tables

INTRODUCTION

This library was designed with the objective of transforming image datasets into query-able SQL tables. A solution like this significantly simplifies the task of extracting meaningful data from images in lieu of data loaders in data science. The construction of this Python library entailed several stages, such as data extraction, feature extraction, data transformation, memory usage, scalability, and flexibility.

INTERNAL MODULES

Data extraction: The initial step of the process, data extraction, is performed by a loader capable of reading a multitude of image formats. These include JPEG, PNG, TIFF, BMP, and more. This feat was achieved by integrating the capabilities of two existing libraries, PIL (Pillow), and OpenCV. Both individual images and directories comprising subdirectories of images can be navigated by the loader, which systematically traverses the file tree.

Feature extraction: After successful data extraction, the next stage involves the extraction of meaningful features. This task is performed by feature extractors equipped to handle basic and complex features. Simple features include color histograms, texture, and shape, while complex ones encompass edges and corners. Moreover, deep learning features derived from pre-trained models are incorporated. OpenCV is the primary tool utilized for simple feature extraction, but when paired with scikit-image, advanced features are effectively extracted.

Data transformation: Subsequent to the extraction of features, they are converted into an SQL format, which enables query-ability. Libraries such as SQL3Lite and psycopg2 are utilized to create SQL tables and to insert data into them. This transformation allows the initially unstructured image data to be stored and queried in a structured manner, facilitating a more efficient data handling process.

IMPLEMENTATION STATUS

At the current stage of development, essential modules of the Python library have been successfully implemented. The Feature Extractor, BLOB Image Storage, and SQL Transformation are primary components now functioning effectively. The refactored Feature Extractor module is capable of retrieving dimensions, color channels, edges, color histograms, corners, etc. from a variety of image formats.

The BLOB Image Storage module includes functions capable of transforming numpy arrays into BLOBs for PostgreSQL and SQLite, given that most images and features are stored as arrays.

Two versions of the SQL Transformation module have been developed: a lite version that generates SQL statements as text and stores them as .sql files, and a sophisticated version that uses psycopg2 or sqlite3 to directly operate on database systems.

IMPLEMENTATION BREAKDOWN

The ImageFeatureDataset class is initialized by its constructor with the database name or database connection string, with the method create_tables() subsequently invoked to ensure the creation of the requisite tables in the database. A context manager method, connect(), is employed for managing the database connections, where a connection to the specified database is opened, and a cursor is yielded for executing SQL statements. After the execution process, the connection is ensured to be closed.

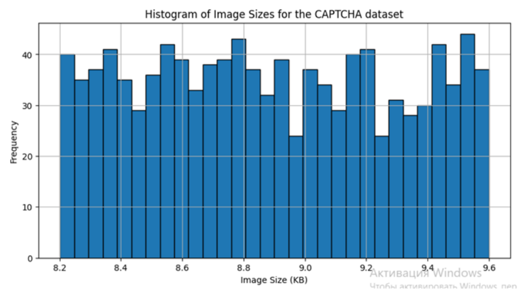
The method create_tables() is used to create two tables, namely 'images' and 'features', in the database, provided they do not already exist. For the input image, a color histogram is computed in the HSV color space and returned as a 1D array by the compute_color_histogram() method. Edge detection in the input image is carried out using the Canny edge detection algorithm by the detect_edges() method, and a binary image with detected edges is returned. Similarly, the detect_corners() method is utilized for detecting corners in the input image using the Shi-Tomasi corner detection algorithm, returning a binary image with the detected corners.

For an image path input, the insert_image_and_features() method computes the image's features (color histogram, edges, corners) and inserts them into the database. The update_image_and_features() method operates similarly, but recomputes the features of the image and updates them in the database. The delete_image_and_features() method is used to remove the specified image's record and its associated features from the database. The paths of all images stored in the database are retrieved by the get_all_images() method, while the get_image_and_features() method retrieves the features of the specified image from the database. The numpy_array_to_blob() and blob_to_numpy_array() methods are used to convert a numpy array into a binary format that can be stored in the database and vice versa, with the latter requiring the original array's shape if it was not a 1D array.

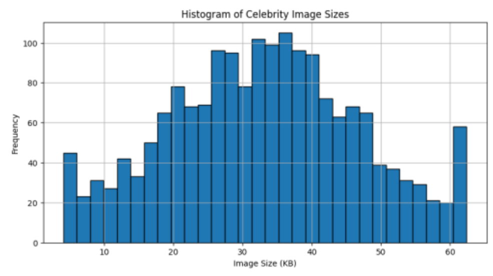
TEST DATA INSIGHTS

The data acquisition strategy involves sourcing three distinct image datasets of varying sizes from Kaggle, for benchmarking scalability. By sorting datasets by size on Kaggle, a suitable selection can be made to test the library's performance across a spectrum of dataset volumes. A proportional growth in execution time is expected relative to the size of the dataset, illustrating the scalability of the library. Insightful observations are offered by the image size histograms from the three image datasets on the appropriate way of processing the datasets by the library. In the majority of scenarios (excluding smaller datasets such as the Captcha Dataset), it's not feasible to load the entire dataset into memory at once.

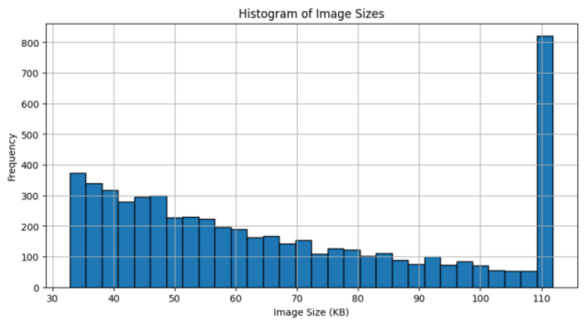
CAPTCHA Images:



Celebrity Face Image Dataset:



WeedCrop Image Dataset:



ANALYSIS

Memory usage and scalability: An essential aspect of the assessment of this Python library is the analysis of its memory usage. Python's 'memory-profiler' library is employed as the primary benchmark in this regard. The library operates chiefly in RAM, hence, it is crucial to evaluate the amount of memory utilized during its operation. The scalability of the library is another pivotal aspect that warrants examination. The library's performance in relation to the increasing dataset size is investigated to understand whether the speed of processing remains proportional as the volume of data grows. Python's 'time' library is employed for the logging of corresponding timestamps, enabling the evaluation of this metric.

Investigative work: Throughout the library's development, significant emphasis has been placed on rigorous analysis. The library's memory usage, deemed as the primary Key Performance Indicator, has been closely monitored using Python's 'memory-profiles' library.A comprehensive survey is being conducted across various platforms, such as blogs, web forums, and social media, to identify the necessary image-processing features for the library. Factors under examination include the time and resources required to retrieve each feature, enabling a more nuanced understanding of the library's performance and its potential areas of improvement.The discovery of a wide range of feature engineering techniques during the feature selection process led to the exploration of incorporating a Feature Selector Module. Recursive Feature Elimination and Sequential Feature Selection are currently under scrutiny to be potentially included in the library.Lastly, defining the scope of the library has been an ongoing effort. A key part of this involves discerning between datasets better suited to transformation into relational schemas rather than NoSQL ones.

Flexibility: The flexibility of this Python library is measured by assessing the variety of image formats and feature types it can manage in comparison to manual methods. The aim is to support most popular image formats employed in conventional Computer Vision. This attribute contributes to the adaptability of the library, making it an ideal tool for a wide range of image processing tasks.