Student: Elshan Naghizade

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

Date: 3 July, 2023

MEASUREMENT STRATEGIES

Profiling memory usage assists in identifying potential memory leaks or inefficient data handling that could potentially hamper the performance of the software. Python's 'memory-profiler' library is an effective tool for this purpose, offering precise measurements of memory consumption during the execution of the software.

To initiate memory profiling, the 'memory-profiler' module must first be imported into the software. I would then decorate the functions with the '@profile' decorator. I would likely begin by profiling the functions responsible for reading and loading image datasets, since these operations are inherently memory-intensive due to the large size of image files. Additionally, the functions that are responsible for feature extraction should also be profiled as they might be computationally heavy and could lead to high memory usage. The functions that convert the extracted features into SQL format, and manage SQL tables also need to be profiled for memory usage. These operations often involve the creation, handling, and disposal of large data structures, which could lead to significant memory consumption if not properly managed.

To execute the script with the memory profiler, I'll use the 'mprof run' command, followed by the script name in the command line. This would generate a .dat file containing the memory usage information of the program.

By examining the numerical memory usage data, I will be able to identify any functions or operations that consume a disproportionate amount of memory. Potential solutions to reduce memory usage could include modifying data structures, avoiding unnecessary data duplication, or employing batch-by-batch processing techniques where possible.

Finally, it's crucial to repeat this process as many times as necessary, particularly after making changes aimed at reducing memory consumption. This iterative approach ensures that memory usage is continuously monitored and optimized, enhancing the performance and scalability of the software.

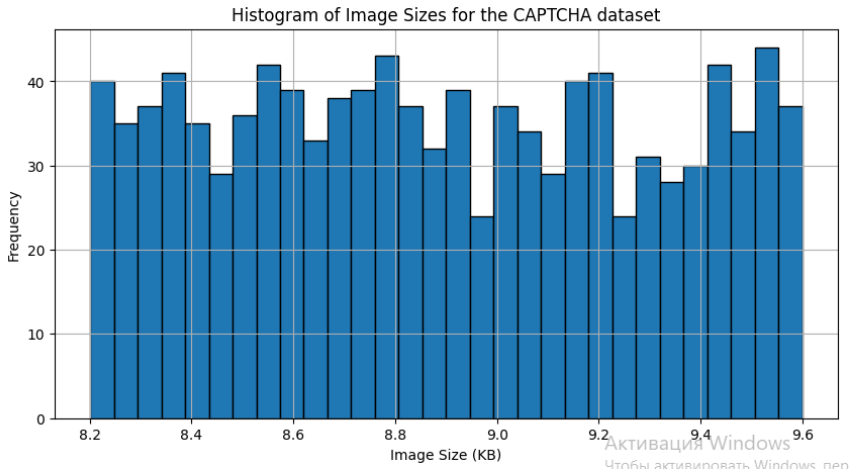
STATISTICAL ANALYSIS

Analyzing particular datasets is out of the scope of my project since I am building a python library which operates in lieu of data-loaders for Computer Vision projects.

VISUALIZATION

CAPTCHA Images: https://www.kaggle.com/datasets/fournierp/captcha-version-2-images

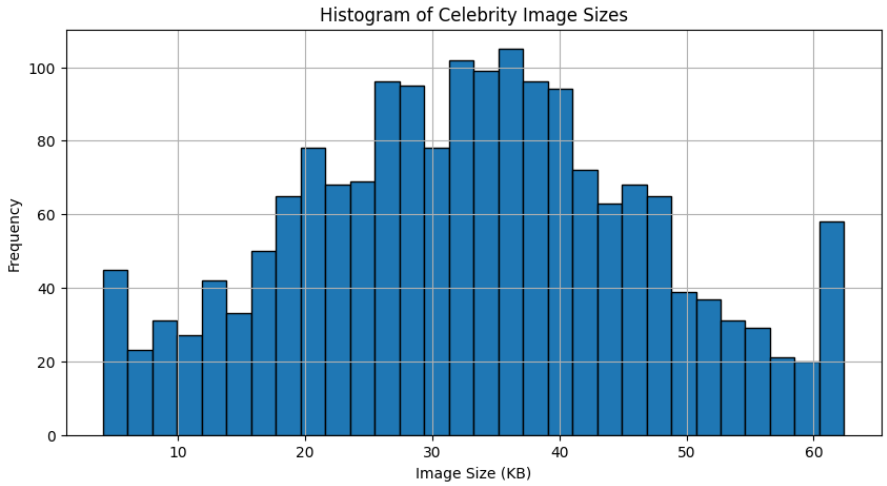
Here is the image size histogram for the Captcha dataset:



Celebrity Face Image Dataset:

https://www.kaggle.com/datasets/vishesh1412/celebrity-face-image-dataset

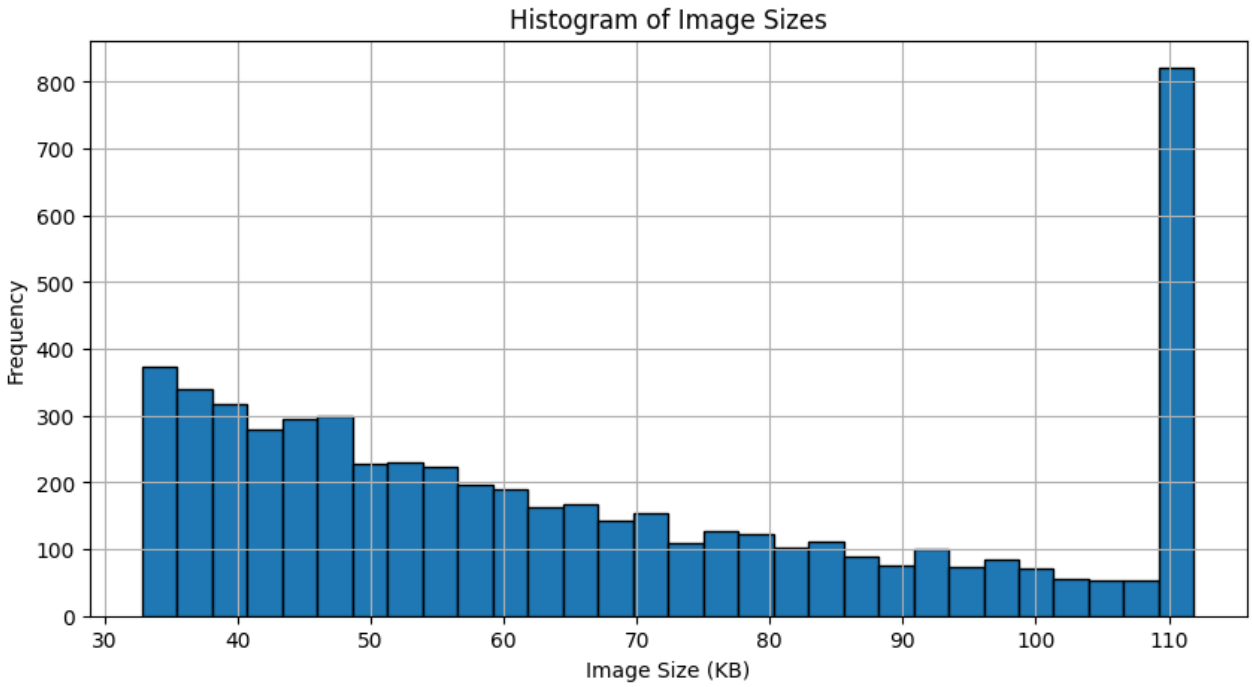
Here is the image size histogram for the Celebrity Face dataset:



WeedCrop Image Dataset:

https://www.kaggle.com/datasets/vinayakshanawad/weedcrop-image-dataset

Here is the image size histogram for the WeedCrop dataset:



VISUAL STORYTELLING

Image size histograms for the 3 image datasets provide a crucial insight into how the datasets should be processed by the library since for most of the cases (excluding small datasets, i.e. Captcha Dataset) the dataset in its entirety can not be loaded into memory at once.

There are two feasible approaches to tackle the issue at hand:

1. Use batch-by-batch loading – Having a separate parameter to specify the size of the batch to be processed at once up until the whole dataset is loaded into a database.

2. Use storage-based memory emulation – This method is much harder to implement, however, Hadoop-based solutions can facilitate the development process.

Taking into account the time limitations of the course the first approach was selected.