Introduction

This work is around the intricate process of flower detection utilizing the SIFT method. The initial phase involved identifying a suitable dataset that aligns with the objectives of the study. Subsequently, a thorough exploration ensued, leading to the development of a robust model tailored for flower detection.

The culmination of this endeavor involved testing to assess the efficacy and performance of the newly devised model in the context of flower recognition.

1 Data

The project uses a Flowers dataset obtained from Kaggle. In this dataset, there are 16 types of flowers in different folders each type on average has 900 pictures.

In Figure 1 is possible to see how the data set is subdivided.

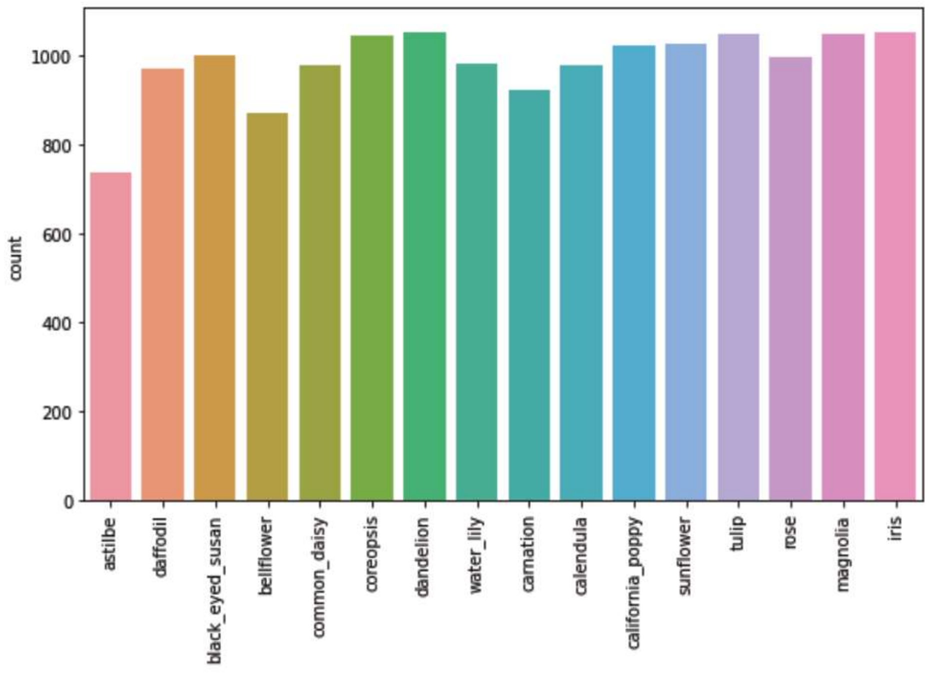


Figura 1: Dataset flowers

2 Proposed Solution: Sift Method

In the development of the program, the chosen approach for image detection and comparison was the SIFT method. SIFT is selected for its ability to handle variations in scale, rotation, and illumination, providing a reliable means to identify distinctive features within images.

The SIFT (Scale-Invariant Feature Transform) method is a robust and widely used technique for detecting and describing distinctive features in images. It's known for its ability to handle scale variations, rotation, and changes in illumination, making it suitable for a variety of computer vision tasks, including image matching, object recognition, and image stitching.

3 Program Structure

The program adopts an object-oriented approach, organized into distinct Python files, each corresponding to a specific class.

The primary classes, outlined in Figure 2, form the backbone of the program.

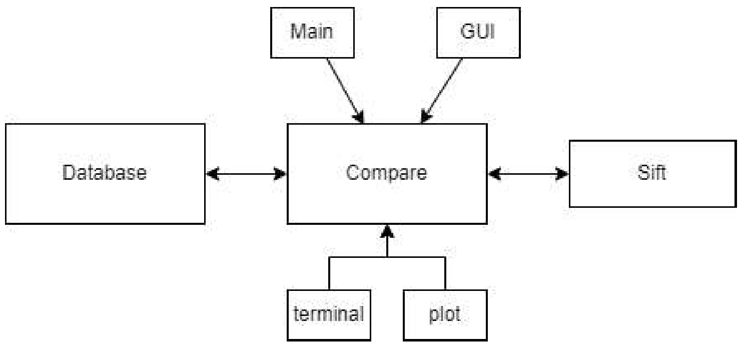


Figura 2: Code Structure

A *Database* represents a collection of folders, each containing images of a specific flower type. The *Database* class facilitates the retrieval of these images for further processing.

The *Compare* class leverages the paths obtained through the *Database* class to invoke functions within the *SIFT* class. This process involves comparing the test image against the database images and determining the number of common points. After a comprehensive comparison, the program identifies the flower type, providing predictions for the test image. The results are then displayed in both the terminal and visualized through plots.

3.1 Comparison Process

During the comparison process, the SIFT algorithm identifies key points and descriptors in the test image and compares them with those in the training set. Matching key points contributes to a similarity score, reflecting the likeness between the test image and each training image.

The similarity score is calculated based on the number of matched key points between the test image and each image of the database. For each type of flower is sum all the matches obtained, and now the most similar folder, which will correspond to the flower of the test image.

4 Program Flow

The program is characterized by two distinct user cases, offering versatility in its utilization.

The Database path is already saved in the program, but on some PC could not work therefore may need to be changed before running the code.

4.1 Training

The first user case is applied to training. Run the code in the file *main.py* processes a sequence of functions in different classes and in short is in the next figure **??**.

The first user case, focused on training, the execution of the code is done in the *main.py* file and involves a series of functions distributed across various classes, as illustrated in Figure 3.

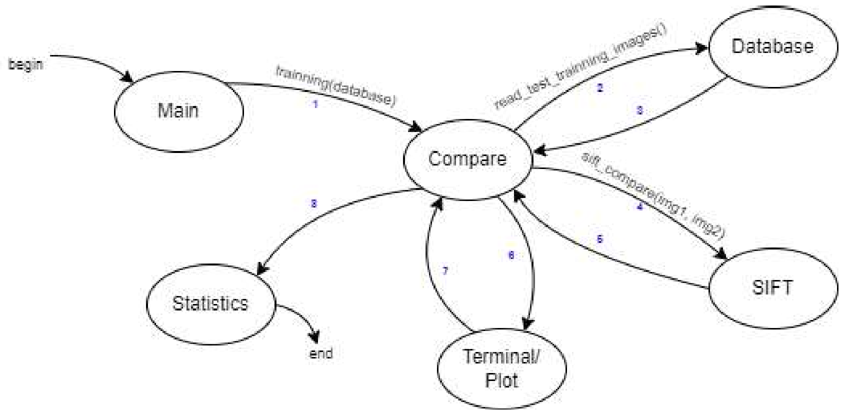


Figura 3: User Case Training

The program leverages the database path, undertaking a crucial step of dividing the dataset into training and testing sets. This segmentation is controlled by two key parameters: the number of images designated for both training and testing purposes per flower type and the number of test images allocated to each flower type.

A default test scenario utilizes merely 100 images for training and one image for testing per folder. This approach is necessitated by the time-intensive nature of the SIFT method during the training phase, reminding the need for a significant reduction in parameters.

The *Database* class ensures that the program optimally prepares the data for subsequent analyses. Subsequently, in the *Compare* class, each test image is systematically compared to the training dataset. Multiple results are generated through these comparisons, enabling the derivation of insightful statistics and identification of the folder with the most analogous path.

The culmination of this process involves a comparison of the aggregated results with the expected outcomes, providing a comprehensive evaluation of the program's performance.

4.2 User Interface

The other user case is run by the file *gui.py*. It starts a GUI interface and is processed like the Figure 4.

The alternative user case is initiated through the execution of the *gui.py* file. This user case involves the activation of a graphical user interface (GUI), as depicted in Figure 3.

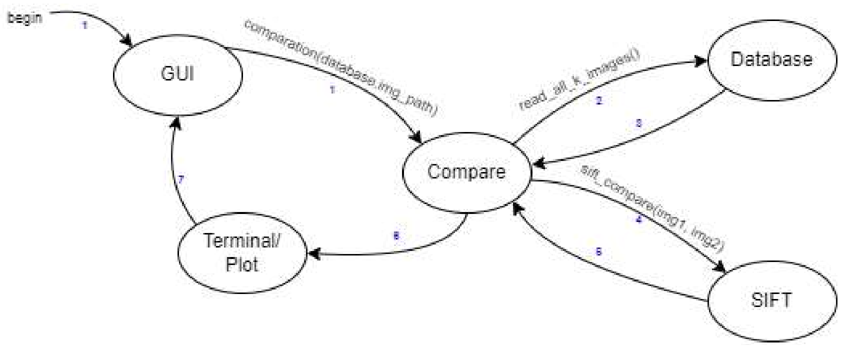


Figura 4: User Case User

Upon execution, the program launches a GUI interface, as illustrated in Figure 5.

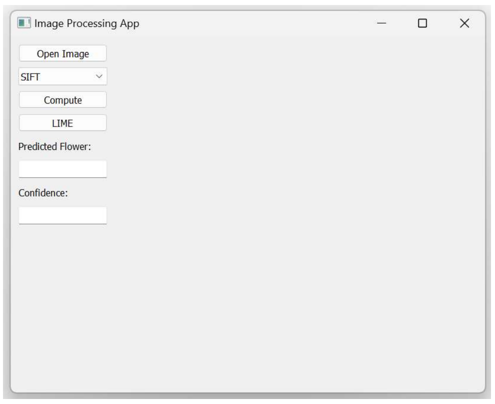


Figura 5: User Case User

The interface provides various button clicks:

- 'Open Figure': The user can select the figure for testing.
- 'Choose Approach': Permits the user to specify the approach to use. In this instance, the default is set to *SIFT*.
- 'Compute': Initiates the code execution. The program conducts a comparison between the chosen image and a predetermined number of images from the database, specifically 100 images, mirroring the train-test split defined previously.
- 'Prediction': Displays the outcome after the code execution.

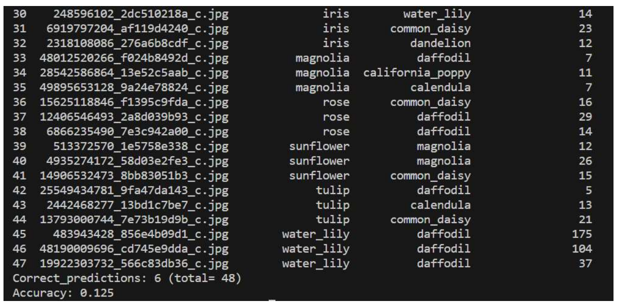
These interactive features enhance the usability of the program, enabling users to select figures, and individuals, and initiate computations effortlessly through a user-friendly graphical interface.

5 Result Analysis

To analyze the results, it uses the training user case and it performs statistical analyses on the similarity scores. We compute the average similarity scores for each flower type, providing insights into the overall matching patterns.

Later, the accuracy is calculated by comparing the predicted flower types with the expected results.

The formula for accuracy is (Number of Correct Predictions / Total Number of Predictions) * 100.



```
30    248596102_2dc510218a_c.jpg           iris        water_lily           14
31   6919797204_af119d4240_c.jpg           iris      common_daisy           23
32   2318108086_276a6b8cdf_c.jpg           iris         dandelion           12
33  48012520266_f024b8492d_c.jpg       magnolia          daffodil            7
34  28542586864_13e52c5aab_c.jpg       magnolia  california_poppy           11
35  49895653128_9a24e78824_c.jpg       magnolia         calendula            7
36  15625118846_f1395c9fda_c.jpg           rose      common_daisy           16
37  12406546493_2a8d039b93_c.jpg           rose          daffodil           29
38   6866235490_7e3c942a00_c.jpg           rose          daffodil           14
39    513372570_1e5758e338_c.jpg      sunflower          magnolia           12
40   4935274172_58d03e2fe3_c.jpg      sunflower          magnolia           26
41  14906532473_8bb83051b3_c.jpg      sunflower      common_daisy           15
42  25549434781_9fa47da143_c.jpg          tulip          daffodil            5
43   2442468277_13bd1c7be7_c.jpg          tulip         calendula           13
44  13793000744_7e73b19d9b_c.jpg          tulip      common_daisy           21
45    483943428_856e4b09d1_c.jpg     water_lily          daffodil          175
46  48190009696_cd745e9dda_c.jpg     water_lily          daffodil          104
47  19922303732_566c83db36_c.jpg     water_lily          daffodil           37
Correct_predictions: 6 (total= 48)
Accuracy: 0.125
```

Figura 6: Results

The Figure 6 shows the result obtained by training of 7 images and 3 images of test for each folder. Unfortunately, the accuracy was to small resulting in a wrong predicted values

6 Problems

One of the main problems that had during the code development was testing a huge quantity of pictures. The processing of the code is too slow because the program needs to compare the test image with all the images from the dataset. One of the solutions that I found was to create a parameter and just test k numbers of images of each folder each time. Another problem was the accuracy which was too small. During the process of creating the code was no time to optimize the results and obtain better accuracy.

7 Future improvements

The program done at the end had a lower score, which is not so good. Was trying many ways to improve the code, but just using the sift method will be not possible. To improve the code will need to improve the processing time and combine it with another method.

Conclusion

Summary of Findings: In conclusion, the experiment reveals that the SIFT method did not identify the flower images too well. But in the end was possible to understand better how to use and implement the SIFT method.