Report Kernel Rebooters: Vision Division

Marco Carraro
marco.carraro.23@studenti.unipd.it

Luca Pellegrini
luca.pellegrini.6@studenti.unipd.it

Francesco Vezzani
francesco.vezzani.1@studenti.unipd.it

February 14, 2026

1 Introduction

The overall goal of this project is to build a complete computer vision pipeline that can classify flower images and support further analysis tasks, such as health-state estimation and performance evaluation. In practical terms, the system is designed around a clear workflow: images are loaded from a structured dataset, visual descriptors are extracted, test images are compared against training references, and final predictions are produced.

TODO...

2 Preprocessing

TODO

3 SIFT

TODO

4 SURF

TODO

5 ORB

TODO

6 Template Matching

TODO

7 HOG

7.1 Objective

The HOG branch was introduced to add a global shape-and-gradient based descriptor that is easy to interpret and relatively stable in many real-world image conditions. The idea is simple: each test image is converted into one feature vector, then compared with feature vectors extracted from training images. The predicted class is taken from the nearest training sample.

From an engineering perspective, the objective was not to create a heavily optimized model, but to produce a transparent baseline that is easy to debug, easy to validate, and fully consistent with the project architecture.

7.2 Code Structure Decision

HOG logic is split into:

- `HOGExtractor` (`include/hog.h`, `src/hog.cpp`) for low-level feature extraction and distance computation.
- `hog(...)` wrapper (`src/matching.cpp`) for dataset-level loop and prediction print.

This separation is important because it keeps responsibilities clean: `HOGExtractor` handles only algorithmic operations, while `hog(...)` handles dataset traversal and prediction flow. As a result, `main.cpp` remains focused on orchestration, and the module can be reused or replaced with minimal impact on the rest of the system.

7.3 Feature Extraction Choices

- Input image is converted to grayscale if needed.
- The image is resized to a fixed window (`64x128`).
- HOG parameters are basic OpenCV defaults: block size `16x16`, block stride `8x8`, cell size `8x8`, 9 bins.

The fixed resize step is a key practical decision: HOG vectors can only be compared directly when they share the same dimensionality. Using a fixed window gives deterministic descriptor length and avoids shape-dependent edge cases in matching.

7.4 Matching Choices

- For each test descriptor, all train descriptors are scanned.
- Similarity score is Euclidean distance (L2).
- Predicted label is the train sample with minimum distance.

This results in a straightforward nearest-neighbor baseline:

$$\hat{y}(x) = \arg\min_i \|h(x) - h(x_i^{train})\|_2$$

where $h(\cdot)$ is the HOG descriptor.

The benefit of this choice is interpretability: every prediction can be traced back to one specific training sample and one explicit distance value, which is very useful during qualitative inspection.

7.5 Robustness Choices

- Empty input image check.
- Empty or incompatible descriptor check before matching.
- Sample is skipped if descriptor extraction fails.

7.6 Trade-off

The implementation is intentionally simple and readable, but the exhaustive comparison step has cost $O(N_{test} \cdot N_{train})$. This is acceptable as a baseline and for medium dataset sizes, but it is a known scalability limitation for larger datasets.

8 BoW

8.1 Objective

While HOG focuses on global gradient structure, BoW was introduced to capture local visual patterns through keypoints. The goal is to convert a variable number of local ORB descriptors into a fixed-size global representation, so images can be compared in a compact and uniform way.

In other words, BoW bridges local detail and global matching: it keeps the discriminative power of local descriptors but produces one standardized vector per image.

8.2 Code Structure Decision

BoW logic is split into:

- `BoWExtractor` (`include/bow.h`, `src/bow.cpp`) for vocabulary building, histogram extraction, and histogram distance.
- `bow(...)` wrapper (`src/matching.cpp`) for train/test loop and prediction print.

This separation follows the same architectural rule used across the project: core algorithm in an extractor class, orchestration in a lightweight wrapper. This improves readability and allows future experiments (different vocabularies, different local features, different distance metrics) without touching application flow.

8.3 Pipeline Choices

The implemented BoW pipeline is:

1. extract ORB descriptors from all train images,
2. convert descriptors to `CV_32F`,
3. run k-means to build a visual vocabulary (default size: 20 words),
4. assign each descriptor to the closest visual word,
5. build one histogram per image and normalize it,
6. compare test and train histograms with L2 distance.

This pipeline is intentionally classical and explicit. Every stage can be inspected independently (descriptors, vocabulary, histogram), making debugging and incremental improvement easier for the team.

8.4 Why ORB + K-means

- ORB is already present in the project and is computationally light.
- K-means gives a direct and standard way to form visual words.
- Fixed vocabulary keeps the implementation deterministic and easy to tune.

Choosing ORB also reduces integration friction, since ORB-based utilities were already available and familiar in the codebase. K-means, despite being simple, provides a clear semantic interpretation: each cluster center is treated as a visual word.

8.5 Histogram and Matching Formula

Each image is represented by a normalized histogram $b(x) \in R^K$, where K is vocabulary size. Prediction follows:

$$\hat{y}(x) = \arg\min_i \|b(x) - b(x_i^{train})\|_2$$

Normalization reduces sensitivity to raw keypoint count differences between images, so comparison focuses more on visual word distribution than on absolute descriptor count. This is especially useful when images produce very different numbers of detected keypoints.

8.6 Robustness Choices

- Vocabulary availability check before extraction.
- Empty descriptor/histogram checks.
- Skip logic when extraction fails for one image.

8.7 Trade-off

The current implementation favors clarity over optimization:

- vocabulary is rebuilt at each run,
- nearest-neighbor matching is exhaustive,
- no TF-IDF or advanced scoring yet.

These trade-offs are deliberate at this stage: the code remains concise and easy to reason about, and it provides a reliable baseline before introducing acceleration or more advanced weighting schemes.

9 Flower Health Detection

Section reserved for flower health detection details (to be completed).

10 Performance Evaluation

Section reserved for metrics, confusion matrix, and timing analysis (to be completed).

11 Conclusion

In this iteration, HOG and BoW were integrated as simple and modular baselines. The main design decision was to keep a consistent extractor interface and move algorithm-level loops outside `main.cpp` into a dedicated matching module.

References / Links

- Project repository: `https://github.com/kekko7072/Final_Project_Kernel_Rebooters/tree/main`
- OpenCV HOGDescriptor documentation: `https://docs.opencv.org/4.x/d5/d33/structcv_1_1HOGDescriptor.html`
- OpenCV ORB documentation: `https://docs.opencv.org/4.x/db/d95/classcv_1_1ORB.html`
- OpenCV kmeans documentation: `https://docs.opencv.org/4.x/d1/d5c/tutorial_py_kmeans_opencv.html`