Dense SIFT BoVW Image Classification: Function-level Breakdown and Pipeline Overview

Anonymous CVPR submission

Paper ID ****

Abstract

We dissect an enhanced Bag-of-Visual-Words (BoVW) image-classification pipeline implemented in `version3.py`*. Instead of reporting empirical results, this report focuses on* ***(i)*** *the role of each major function, and* ***(ii)*** *how these functions interoperate to form a coherent training-to-inference workflow.*

1. Pipeline at a Glance

The pipeline can be summarised in five sequential stages:

1. **Dense SIFT Extraction**
2. **PCA + KMeans Vocabulary Learning**
3. **BoVW Histogram Encoding**
4. **Feature Standardisation**
5. **Linear SVM Training / Prediction**

Each stage corresponds to a dedicated function set in `version3.py` (Table **??**).

2. Key Functions

2.1. Feature-related

`extract_dense_sift(img)` Applies CLAHE, samples keypoints on a 4 px grid across four scales, and returns $d = 128$-dimensional SIFT descriptors with ℓ_2 normalisation.

`build_vocabulary(training_dir,...)` Aggregates descriptors from all classes, standardises them, projects to 128-D PCA space, and learns a 1000-word MiniBatchKMeans codebook. It outputs the fitted scaler, PCA model, and KMeans object.

`extract_bow_features(img, scaler, pca, vocab)` Converts an arbitrary image into a BoVW histogram by: (1) dense SIFT, (2) PCA projection, (3) FAISS nearest-centre lookup, (4) ℓ_1 normalisation.

2.2. Data Loading

`load_training_data()` Iterates over labelled folders, calls `extract_bow_features`, and yields a feature matrix $\mathbf{H}_{\text{train}} \in \mathbb{R}^{n\times 1000}$ and label vector.

`load_test_data()` Mirrors the above but records file names instead of labels.

2.3. Model Training and Evaluation

`train_svm_classifier()` Performs a standardisation pass, prints class distribution, runs a 5-fold grid search over Linear-SVM hyper-parameters, and finally fits the best One-Vs-Rest estimator.

`predict_and_save()` Applies the feature scaler, obtains decision values / predictions, prints sanity-check statistics, and writes ordered results to disk.

3. Putting It Together: `main()`

Listing 1 outlines the chronological invocation order and data objects exchanged between functions.

1. **Vocabulary Building** — `build_vocabulary`
2. **Training Feature Matrix** — `load_training_data`
3. **Classifier Training** — `train_svm_classifier`
4. **Model Serialisation** — via `pickle`
5. **Test Feature Matrix** — `load_test_data`
6. **Inference** — `predict_and_save`

4. Design Rationale

Dense vs. Sparse SIFT. Dense sampling guarantees uniform coverage, crucial for fine-grained classes.

PCA Before Clustering. A 128-D projection preserves most variance while lowering memory and speeding up both KMeans and FAISS search.

```
scaler, pca, kmeans = build_vocabulary()
X_train, y = load_training_data(scaler, pca, kmeans)
clf, feat_scaler   = train_svm_classifier(X_train, y)
pickle.dump((...), open('model.pkl','wb'))
X_test, fnames     = load_test_data(scaler, pca, kmeans)
predict_and_save(clf, feat_scaler, X_test, fnames)
```

Figure 1. Core control flow (pseudo-code).

FAISS Quantisation. Compared with brute-force search, FAISS scales to millions of descriptors with negligible loss in assignment accuracy.

Linear SVM Choice. Empirically sufficient for high-dim BoVW histograms; training remains fast with the `dual` optimisation mode.

5. Conclusion

Each function in `version3.py` encapsulates a self-contained step of the classical BoVW pipeline. Understanding their interfaces clarifies how data flow and hyper-parameters interact, enabling straightforward modification or replacement (e.g., swapping SIFT for ORB, or Linear-SVM for logistic regression).

References

[1] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *IJCV*, 2004.

[2] J. Johnson and H. Zhang. Billion-scale similarity search with GPUs. *IEEE TKDE*, 2019.

[3] F. Pedregosa *et al*. Scikit-learn: Machine Learning in Python. *JMLR*, 2011.