Assignment #2

Prof. Hanbyul Joo
M1522.001000, Computer Vision

Assigned: April 20, 2025
Due: March 5, 2025, 11:59 PM

0 Instruction

In this assignment, you will implement scene recognition with BoW, Eigenface and solve related problems.

- **Submission Platform**: All homework must be submitted electronically on eTL.
- **Collaboration Policy**: Discussions with peers are encouraged, but you must solve the problems and write up the solutions independently.
- **Individual Assignment**: Each student is required to submit their own individual report and code.
- **Plagiarism Policy**: Do **not** copy code or reports from others. Any form of plagiarism may result in a score of zero.
- **Coding Requirements**:
  - Use **Python** for all programming tasks.
  - You are **not allowed** to use any libraries or functions other than those provided in the skeleton code, in order to ensure consistency and fairness in grading.
  - Also, for the same reason, changing the format of the skeleton code is **prohibited**.
- **Reporting**:
  - Answers must be **clear, unambiguous**, and **supported by experimental results** (e.g., images, plots, brief quantitative analysis).
  - Only PDF submissions compiled using LaTeX (e.g., via Overleaf or other LaTeX tools) will be accepted. No specific template will be provided.
- **Submission**:
  - Submit the zip file as the following folder and file structure.

```
HW2_{YOUR_NAME}_{YOUR_STUDENT_ID_NUMBER}.zip
├── 1_scene_recognition
│   ├── scene_recognition.ipynb
│   └── outputs
│       ├── bow_knn_confusion_acc.png
│       └── bow_svm_confusion_acc.png
├── 2_eigenface
│   ├── eigenface.ipynb
│   ├── eigenface_test.png
│   ├── person1_1.png
│   ├── person1_2.png
│   └── person2_1.png
└── HW2.pdf (your report)
```

  - Make sure that **all outputs are preserved**.
  - All images in the report should be **reproducible**.
  - TAs will primarily check if your code is **fully reproducible**.

- **Questions**: We will only respond to questions posted on the eTL Q&A board.

1 Scene Recognition (50 Points)

The goal of this assignment is to build a set of visual recognition systems that classify the scene categories. The scene classification dataset consists of 15 scene categories including office, kitchen, and forest. The system will compute an image representation (bag-of-word visual vocabulary) and predict the category of each testing image using the classifiers (KNN and SVM) built on the training data. The Python skeleton code (`scene_recognition.ipynb`) and dataset(`scene_classification_data`) are provided on the eTL.

- **Do not modify** the skeleton code. You are only allowed to edit the parts marked as **To do**.
- There is no need to submit dataset(`scene_classification_data`).

Instructions

(a) (30 Points) Implement Code:
In `scene_recognition.ipynb`, you need to implement the following **8** functions:

- `extract_dataset_info(data_path)`
  - A function to preprocess the dataset for training and testing.
  - **Output:** `label_classes` is a list of labels. `img_train_list` and `img_test_list` are lists of image paths. `label_train_list` and `label_test_list` are lists of labels for images.
  - **Hint:** You can use `Path` and `PureWindowsPath` from `pathlib`.
- `compute_dsift(img)`
  - Given an image, instead of detecting key points and computing sift descriptor, this function directly compute sift descriptor on a dense set of locations on image.
  - **Output:** `dense_feature` is a collection of sift features whose size is `n`×128. `n` is total number of locations to compute sift features on `img`.
  - **Hint:** You can use sift related functions from `opencv` for computing sift descriptor for each location.
- `predict_knn(feature_train, label_train, feature_test, k)`
  - KNN classifier to predict the label of the test data.
  - **Output:** `label_test_pred` is a n_{te} vector that specifies the predicted label for the testing data.
  - **Hint:** You can use `NearestNeighbors` for KNN
- `build_visual_dictionary(dense_feature_list, dic_size)`
  - Given a list of dense sift feature representation of training images, you will build a visual dictionary made of quantized SIFT features.
  - **Output:** `vocab` lists the quantized visual words whose size is `dic_size`×128.
  - **Hint:** You may start `dic_size`=50. You can use `KMeans` function imported from `sklearn.cluster`.
  - **Note:** It may take quite some time, so please don't be alarmed.
- `compute_bow(feature, vocab)`
  - Give a set of SIFT features from an image, you will compute the bag-of-words feature. The BoW feature is constructed by counting SIFT features that fall into each cluster of the vocabulary. The histogram needs to be normalized such that BoW feature has a unit length.
  - **Output:** `bow_feature` is the bag-of-words feature vector whose size is `dic_size`.
  - **Hint:** You can use `NearestNeighbors` to find the closest cluster center.
- `classify_knn_bow(label_classes, label_train_list, img_train_list, label_test_list, img_test_list)`
  - Given BoW features, you will combine `build_visual_dictionary`, `compute_bow`, and `predict_knn` for scene classification.

- **Output:** `confusion` is a 15×15 confusion matrix and `accuracy` is the accuracy of the testing data prediction.
- **Note:** The goal is to get your best setup to accuracy of >55%.

- `predict_svm(feature_train, label_train, feature_test)`
  - You will use a SVM classifier to predict the label of the testing data. To decide which of 15 categories a test case belongs to, you will train 15 binary, 1-vs-all SVMs. All 15 classifiers will be evaluated on each test case and the classifier which is most confidently positive "wins".
  - **Output:** `label_test_pred` is a n_{te} vector that specifies the predicted label for the testing data.
  - **Hint:** You can use function `LinearSVC` or `SVC` imported from `sklearn.svm`. When learning an SVM, you have a free parameter 'lambda' (argument `C` in function `LinearSVC` and `SVC`) which controls how strongly regularized the model is. Your accuracy will be very sensitive to lambda, so be sure to test many values.
  - **Note:** `LinearSVC` and `SVC` can do multi-class classification if your input labels have more than 2 classes. However, you **should not** take advantage of that. Instead, you should create binary labels for each of those 15 binary 1-vs-all SVMs.

- `classify_svm_bow(label_classes, label_train_list, img_train_list, label_test_list, img_test_list)`
  - Given BoW features, you will combine `build_visual_dictionary`, `compute_bow`, `predict_svm` for scene classification.
  - **Output:** `confusion` is a 15×15 confusion matrix and `accuracy` is the accuracy of the testing data prediction.
  - **Note:** The goal is to get your best setup to accuracy of >65%.

(b) (20 Points) Write Report:
Write a report analyzing the results. In particular, try adjusting the hyperparameters used in your implementation and analyze how the accuracy varies depending on the settings. The best setting should achieve an accuracy that surpasses the baseline provided above.

Reference

- `https://www-users.cse.umn.edu/~hspark/csci5561_S2021/hw3.pdf`

2 Eigen Face (50 Points)

- **Do not** modify the skeleton code itself. You may only edit the sections marked with **TODO**.
- If you used any additional images for testing, please submit those files as well, and do not clear the outputs in the notebook.
- You are not allowed to import any additional libraries and must use only the libraries that are already imported in the skeleton code.

Instructions

(a) (40 Points) Implement the Code:
In `eigenface.ipynb`, you are required to complete the following `8` sections marked as `TODO`:

- `get_grid_image(images, n_row, n_col)` – (5 points)
  - Implement a function to visualize a batch of images as a grid.
  - **Output:** The shape of `grid_image` should be `n_row*height, n_col*width`.
- `compute_eigenvector(data)` – (5 points)
  Compute eigenvectors and eigenvalues from the given dataset.
  - **Input:** Dataset of shape `N x feature_dim`
  - **Output:** `eigenvalues` (`feature_dim`), `eigenvectors` (`feature_dim x feature_dim`)
- `Step (2-1, 2-2)` – (5 points each)
  Visualize the top-N and bottom-N eigenvectors as images.
- `decompose_image_pca(image_flatten, eigenvectors)` – (5 points)
  - Implement a function to decompose a flattened image using the provided eigenvectors.
  - **Output:** PCA coefficients and the reconstructed image using those coefficients and eigenvectors.
- `Step (3-2)` – (5 points)
  Reconstruct an image using a reduced number of PCA coefficients (e.g., 100 and 3000) and visualize the reconstructed results.
- `compute_cosine_similarity(img_path1, img_path2)` – (5 points)
  - Implement a function that returns the cosine similarity between two images.
  - **Input:** `img_path1`, `img_path2`, and optionally other parameters.
  - **Hint:** You may modify the function signature to include additional parameters. Consider what inputs are needed to perform the computation.

(b) (10 Points) Write a Report:

- In `Step 1`, two types of normalization were applied to the image data. Explain the effect each normalization method has on the data.
- Describe the role of eigenvectors in relation to the eigenvalue order, and discuss the advantages and disadvantages of using a reduced number of eigen-coefficients.