Computer Vision Lab #9

Implement Visual Place Recognition using Bag of Visual Words

Objective:

To implement a visual place recognition system that identifies the top 10 most similar images to a given query image from a database using the **Bag of Visual Words model**.

Required Tools:

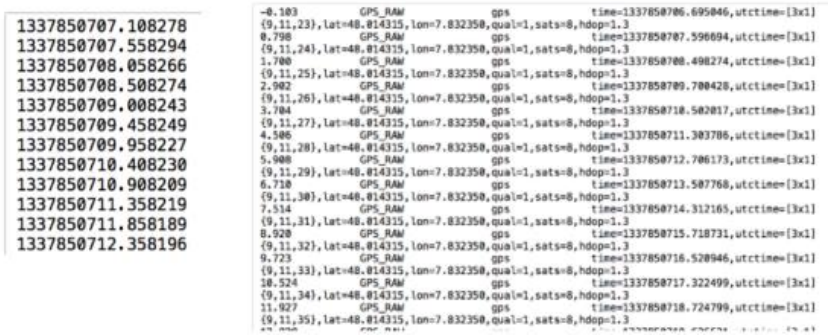
- Python 3.x
- OpenCV library for Python
- scikit-learn library for Python
- NumPy library
- A dataset of images (e.g., Freiburg dataset)

Task 1: Implement a visual place recognition system following these steps

Part 1: Preparation and Setup

1. **Download and Setup Dataset**:
    - Download the Freiburg dataset from the provided link. https://uni-bonn.sciebo.de/s/c2d0a1ebbe575fdba2a35a8033f1e2ab
    - Unzip and store the images in a directory accessible to your Python script.

    Data Examples



2. **Software Installation**:
    - Ensure Python 3.x is installed on your system.
    - Install necessary Python libraries using pip:

```
pip install numpy opencv-python-headless scikit-learn
```

Part 2: Image Loading and Preprocessing

1. **Load Images**:
    - Write a function to read all images from the dataset directory.
    - Convert images to grayscale to simplify processing and improve computation speed.
2. **Feature Extraction**:
    - Use SIFT (Scale-Invariant Feature Transform) to extract keypoints and descriptors from each image.

Part 3: Building the Visual Dictionary

1. **Feature Aggregation**:
    - Combine descriptors from all images into a single matrix.
2. **K-Means Clustering**:
    - Apply k-means clustering to the aggregated descriptors to form **k** clusters; start with **k=1000**.

```
Import the kmeans from
from sklearn.cluster import KMeans
```

    - Each cluster center represents a "visual word" in the BoVW dictionary.

Part 4: Encoding Images

1. **Histogram Encoding**:
    - For each image, create a histogram that counts how many descriptors belong to each visual word.
    - Normalize histograms to account for varying image sizes and descriptor counts.

Part 5: Query Processing

1. **Query Image**:
    - Extract features from the query image(s).
    - Encode these features using the same visual dictionary to get a histogram.
2. **Similarity Search**:
    - Use Nearest Neighbors to find the 10 closest histograms in the dataset. Import it from

      from sklearn.neighbors import NearestNeighbors

    - Use the Euclidean distance as a measure of similarity.

Part 6: Visualization of Results

1. **Output and Visualization**:
    - Display the top 10 similar images.
    - Optionally, generate an HTML file to present these images in a browser for easy viewing.

Example Code Snippets:

```
import cv2
import numpy as np
import os
from sklearn.cluster import KMeans
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import webbrowser
import tempfile

def load_images(image_folder):
    images = []
    for filename in os.listdir(image_folder):
        path = os.path.join(image_folder, filename)
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        if img is not None:
            images.append(img)
    return images

def extract_features(images):
    sift = cv2.SIFT_create()
    descriptors_list = []
    keypoints_list = []
    for img in images:
```

```
            keypoints, descriptors = sift.detectAndCompute(img, None)
            if descriptors is not None:
                descriptors_list.append(descriptors)
                keypoints_list.append(keypoints)
        return keypoints_list, descriptors_list

    def build_codebook(descriptors_list, codebook_size=1000):
        #your code here
        return kmeans


    def encode_features(descriptors_list, kmeans):
        histograms = []
        for descriptors in descriptors_list:
            #your code here
        return histograms

    def find_similar_images(query_histogram, histograms):
            #your code here
        return indices[0]

    def visualize_results(image_paths, indices):
            #your code here


    def main():
        image_folder = 'your image path'
        images = load_images(image_folder)
        _, descriptors_list = extract_features(images)
        kmeans = build_codebook(descriptors_list)
        histograms = encode_features(descriptors_list, kmeans)
        query_histogram = histograms[0]  # Assuming first image is the query
        indices = find_similar_images(query_histogram, histograms)
        image_paths = [os.path.join(image_folder, f) for f in
os.listdir(image_folder)]
        visualize_results(image_paths, indices)

    if __name__ == '__main__':
        main()
```

Task 2: Parameter Optimization:

- Explore the effect of **varying the number of visual words** (i.e., the k in k-means) on the recognition performance. Start with small values and gradually increase to see the impact on accuracy and computation time.
- Experiment with different feature detectors such as ORB or SURF, and compare their effectiveness and efficiency against SIFT.

Task 3: Weighting Visual Words:

- Investigate the use **of Term Frequency-Inverse Document Frequency (TF-IDF) weighting** to give more importance to less frequent visual words that might be more distinctive.