CV_A4 Image Retrieval

2021315782 이진희

1. 데이터 로드 및 전처리

```
def load_sift_features(file_path):
    with open(file_path, 'rb') as f:
        data = np.fromfile(f, dtype=np.uint8)
    num_features = len(data) // 128
return data.reshape((num_features, 128))

def load_cnn_features(file_path):
    with open(file_path, 'rb') as f:
        data = np.fromfile(f, dtype=np.float32)
    return data.reshape((14, 14, 512))
```

load_sift_features(file_path): SIFT 특징을 바이너리 파일에서 읽어와 메모리에 로드합니다. 각 SIFT 특징은 128차원 벡터로 표현됩니다.

load_cnn_features(file_path): CNN 특징을 바이너리 파일에서 읽어와 메모리에 로드합니다. CNN 특징은 14x14x512 크기의 3차원 텐서로 표현됩니다.

```
def average_pooling(features, pool_size=2):
    pooled_features = features.reshape(features.shape[0] // pool_size, pool_size,
                                       features.shape[1] // pool_size, pool_size,
                                       features.shape[2]).mean(axis=(1, 3))
    return pooled_features

def max_pooling(features, pool_size=2):
    pooled_features = features.reshape(features.shape[0] // pool_size, pool_size,
                                       features.shape[1] // pool_size, pool_size,
                                       features.shape[2]).max(axis=(1, 3))
    return pooled_features
```

average_pooling(features, pool_size) 및 max_pooling(features, pool_size): CNN 특징에 대해 각각 average pooling과 max pooling을 적용하여 features를 조합합니다. 이 과정에서 feature의 정보를 보존하고 차원을 줄입니다.

2. VLAD (Vector of Locally Aggregated Descriptors) 계산

compute_vlad(descriptors, kmeans): VLAD 방법을 사용하여 각 클러스터의 중심과 해당 클러스터에 속하는 데이터 포인트 간의 차이를 계산하여 잔차 벡터를 형성합니다. 이러한 벡터는 L2 정규화하여 특징 벡터를 생성합니다.

```
def compute_vlad(descriptors, kmeans):
    k = kmeans.n_clusters
    centers = kmeans.cluster_centers_
    labels = kmeans.predict(descriptors)
    vlad = np.zeros((k, descriptors.shape[1]), dtype=np.float32)
```

```python
    for i in range(k):
        if np.sum(labels == i) > 0:
            vlad[i] = np.sum(descriptors[labels == i] - centers[i], axis=0)

    # Normalize intra-normalization
    vlad = normalize(vlad, norm='l2')
    # Flatten the VLAD vector
    vlad = vlad.flatten()
    # Normalize final VLAD vector
    vlad = normalize(vlad.reshape(1, -1), norm='l2')

    return vlad.flatten()
```

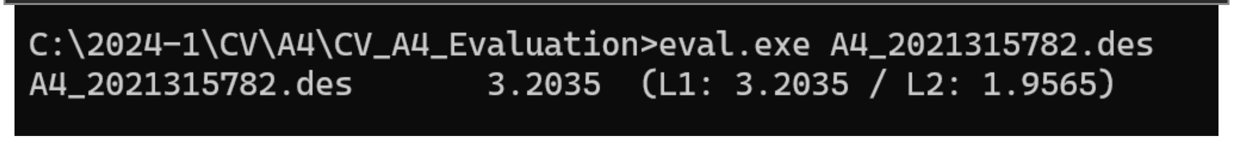
4. Descriptor 생성

PCA와 K-means 클러스터링을 사용하여 각 이미지에 대해 평균 SIFT, 평균 CNN, SIFT VLAD, CNN VLAD를 계산합니다. 이를 결합하여 하나의 Descriptor를 생성합니다.

```python
def compute_descriptors_with_vlad(sift_dir, cnn_dir, num_clusters=60):
    num_images = 2000
    descriptors = []

    kmeans_sift = KMeans(n_clusters=num_clusters, init='k-means++', random_state=0)
    kmeans_cnn = KMeans(n_clusters=num_clusters, init='k-means++', random_state=0)

    scaler_sift = StandardScaler()
    scaler_cnn = StandardScaler()

    pca = PCA(n_components=64)

    all_sift_features = []
    all_cnn_features = []

    for i in range(num_images):
        sift_file = os.path.join(sift_dir, f'{i:04d}.sift')
        cnn_file = os.path.join(cnn_dir, f'{i:04d}.cnn')

        sift_features = load_sift_features(sift_file)
        cnn_features = load_cnn_features(cnn_file)

        pooled_cnn_avg = average_pooling(cnn_features)
        pooled_cnn_max = max_pooling(cnn_features)
        combined_cnn_features = np.concatenate((pooled_cnn_avg, pooled_cnn_max), axis=-1)

        if sift_features.size > 0:
            all_sift_features.append(sift_features)

        if combined_cnn_features.size > 0:
            all_cnn_features.append(combined_cnn_features.reshape(-1, 1024))

    all_sift_features = np.vstack(all_sift_features)
    all_cnn_features = np.vstack(all_cnn_features)

    all_sift_features = pca.fit_transform(all_sift_features)

    all_sift_features = scaler_sift.fit_transform(all_sift_features)
    all_cnn_features = scaler_cnn.fit_transform(all_cnn_features)

    kmeans_sift.fit(all_sift_features)
    kmeans_cnn.fit(all_cnn_features)

    for i in range(num_images):
        sift_file = os.path.join(sift_dir, f'{i:04d}.sift')
        cnn_file = os.path.join(cnn_dir, f'{i:04d}.cnn')

        sift_features = load_sift_features(sift_file)
        cnn_features = load_cnn_features(cnn_file)

        if sift_features.size == 0 or cnn_features.size == 0:
            continue

        pooled_cnn_avg = average_pooling(cnn_features)
        pooled_cnn_max = max_pooling(cnn_features)
        combined_cnn_features = np.concatenate((pooled_cnn_avg, pooled_cnn_max), axis=-1)

        avg_sift = np.mean(sift_features, axis=0)
        avg_cnn = np.mean(combined_cnn_features, axis=(0, 1))

        sift_features = scaler_sift.transform(pca.transform(sift_features))
        combined_cnn_features = scaler_cnn.transform(combined_cnn_features.reshape(-1, 1024))

        sift_vlad = compute_vlad(sift_features, kmeans_sift)
        cnn_vlad = compute_vlad(combined_cnn_features, kmeans_cnn)

        descriptor = np.concatenate((avg_sift, avg_cnn, sift_vlad, cnn_vlad))
        descriptors.append(descriptor)

    descriptors = np.array(descriptors, dtype=np.float32)

    final_descriptors = np.zeros((len(descriptors), 2000), dtype=np.float32)
    for i, desc in enumerate(descriptors):
        if len(desc) > 2000:
            final_descriptors[i] = desc[:2000]
        else:
            final_descriptors[i, :len(desc)] = desc

    return final_descriptors
```

6. 결과 저장

```python
def main():
    base_dir = './features/'
    sift_dir = os.path.join(base_dir, 'sift')
    cnn_dir = os.path.join(base_dir, 'cnn')
    output_file = 'A4_2021315782.des'

    descriptors = compute_descriptors_with_vlad(sift_dir, cnn_dir)

    num_images, dim = descriptors.shape
    with open(output_file, 'wb') as f:
        f.write(struct.pack('i', num_images))
        f.write(struct.pack('i', dim))
        for descriptor in descriptors:
            f.write(struct.pack('f' * dim, *descriptor))
```



```
C:\2024-1\CV\A4\CV_A4_Evaluation>eval.exe A4_2021315782.des
A4_2021315782.des        3.2035  (L1: 3.2035 / L2: 1.9565)
```