A3 report

2020314069

김지민

1. 문제 정의

주어진 feature들의 경우 Image 2000장에 대한 SIFT와 CNN feature로 구성되어 있으며, CNN feature의 형식은 14 x 14 x 512로 구성되어 있고, SIFT feature의 형식은 n x 128로 구성되어 있다. 주어진 feature들을 활용하여 feature descriptor 코드를 작성하고, 이를 바탕으로 Image Retrieval task를 수행한다.

2. 개요

본 코드는 SIFT feature는 사용하지 않고, CNN feature만을 활용하여 task를 수행하였으며, Image 하나에 대한 descriptor의 차원은 2048이다.

3. 코드

```
def cnn_features(path):
    listdir = sorted(os.listdir(path))
    feature = []
    for feat in listdir:
        with open(os.path.join(path, feat), 'rb') as f:
            map = f.read()
        feature.append(np.frombuffer(map, dtype=np.float32).reshape((14, 14, 512)))
    return feature
```

우선 cnn_feature 함수를 이용하여 cnn feature들을 load하여 (14, 14, 512) 형태로 reshape하여 저장하였다.

이후 총 4가지의 연산을 통해 feature descriptor를 제작하였다.

```
def get_GAP(feature):
    GAP = []
    for feat in feature:
        GAP.append(np.mean(feat, axis = (0, 1)))
    return np.array(GAP)
```

우선 get_GAP 함수를 통해서 14x14x512의 feature map에 대하여 global average pooling을 수행하였다. 이를 통해 (2000, 512) 크기의 feature를 획득하였으며 이를 eval.exe를 통해 평가하였을 때, 3.1의 score를 얻을 수 있었다. 즉, GAP 연산이 image의 필터에 대한 reponse를 잘 encoding하였다는 판단을 하여 이를 바탕으로 추가적인 연산들을 진행하였다.

```python
def get_std(feature):
    std = []
    for feat in feature:
        std.append(np.std(feat, axis = (0, 1)))
    return np.array(std)
```

두 번째로 get_std 함수를 통해 feature map의 각 채널에 대한 std 값을 획득하였다. GAP 연산을 통해 얻은 벡터와 concatenation하여 score를 측정해본 결과, 약 3.2로 성능이 향상됨을 확인하였다.

```python
def local_GMP(feature):
    LGmp = []
    for feat in feature:

        feat1 = feat[:7, :7, :]
        feat2 = feat[:7, 7:, :]
        feat3 = feat[7:, :7, :]
        feat4 = feat[7:, 7:, :]
        GMP1 = np.max(feat1, axis = (0, 1))
        GMP2 = np.max(feat2, axis = (0, 1))
        GMP3 = np.max(feat3, axis = (0, 1))
        GMP4 = np.max(feat4, axis = (0, 1))

        all = GMP1 + GMP2 + GMP3 + GMP4
        LGmp.append(all)
    return np.array(LGmp)
```

세 번째로, local_GMP 함수를 통해 14x14x512의 feature map을 2x2 patch로 분할하고, 이에 대해 Global max pooling을 수행하여 patch의 max값에 대한 평균을 취하여 (2000, 512)의 feature를 획득하였다. 전체 Image에 대해서 max pooling 시 다른 feature에 대한 정보가 손실됨을 고려하여 2x2 patch로 분할하여 수행하였다.

```python
def local_GAP(feature):
    LGAP = []
    for feat in feature: #2x2로 나누어서 각각 gap를 구함

        feat1 = feat[:7, :7, :]
        feat2 = feat[:7, 7:, :]
        feat3 = feat[7:, :7, :]
        feat4 = feat[7:, 7:, :]
        GAP1 = np.mean(feat1, axis = (0, 1))
        GAP2 = np.mean(feat2, axis = (0, 1))
        GAP3 = np.mean(feat3, axis = (0, 1))
        GAP4 = np.mean(feat4, axis = (0, 1))

        all = GAP1 + GAP2 + GAP3 + GAP4
        LGAP.append(all)

    return np.array(LGAP)
```

마지막으로, local_GAP 함수를 통해 전체 feature map에 대한 GAP 뿐만 아니라 추가적으로 image를 2x2 patch로 분할하여 GAP를 수행하였고, 위와 같은 연산들을 통

해 최종 descriptor (2000, 2048) file을 생성하였다.

4. 결론

최종 descriptor file을 eval.exe를 통해 score를 측정해본 결과,

```
C:\Users\kimjimin\Downloads\CV_A3_P2_Eval>eval.exe A3_2020314069.des
A3_2020314069.des       3.3130  (L1: 3.3130 / L2: 2.9855)
```

최종적으로 3.3130의 score를 획득하였다.