Introduction to Computer Vision

Assignment #4 Report

2017313008 김태은

```
N = 1000 # num of images
D = 1000 # dimensionality of descriptor
sifts = []  # (1000,n,128)  pre-extracted SIFT features of 1000 images
              # each sift have (n, 128) dim
centers = np.zeros((D, 128), dtype=np.float32)  # 각 클러스터 (D=1000 개) 의 중심 (D,128)
d = np.zeros((N, D), dtype=np.float32)  # descriptor
```

1. Read pre-extracted SIFT & initialize center of clusters

```
sift_path = "./sift/sift100"

for i in range(0, N):
    f = open(sift_path + str("%03d" % i), 'rb')
    data = f.read()
    sift = np.frombuffer(data, dtype=np.uint8)
    sift = np.reshape(sift, (-1, 128))
    sifts.append(sift) # sift 저장
    centers[i] = np.average(sift, axis=0) # 가장 처음의 center 는 각 sift 들(1000 개)의 평균
    f.close()
```

먼저 1000개의 SIFT 파일들을 open하고 read한다. 이 때 각 SIFT feature들은 128 차원의 벡터이다. 총 N(1000)개의 이미지에 대해 SIFT 파일들을 읽어들이고 저장하면 sifts의 dimension은 (1000, n, 128)이 된다. 이 때 n의 값은 각 이미지마다 (SIFT 파일마다) 다른 값이다.

K-means algorithm을 수행하기에 앞서 각 클러스터의 중심을 초기화해야 한다. 클러스터의 개수(D)를 N과 같은 1000개로 잡았기 때문에, 각 이미지(SIFT 파일)의 SIFT feature들의 평균으로 초기화한다.

2. K-means clustering & compute image descriptors

```
# Kmeans clustering
num_epoch=20
for epoch in range(num_epoch):
    new_cluster = []  # 각 SIFT feature 들 (1000 x n 개) 이 어느 클러스터로 분류되었는지 저장
    new_cluster_sum = np.zeros((D, 128)) # 각 클러스터로 분류된 SIFT feature 들의 합
    new_cluster_cnt = np.zeros((D, 1))  # 각 클러스터로 분류된 SIFT feature 들의 개수
```

k-means clustering을 num_epoch 만큼 수행한다.

```
        for i in range(0, N):
            cluster_update= [0] * len(sifts[i])

            for j in range(0, len(sifts[i])):
                dis = np.linalg.norm(sifts[i][j] - centers, axis=1, ord=2)
                min_idx = np.argmin(dis)
                cluster_update[j] = min_idx
                new_cluster_sum[min_idx] += sifts[i][j]
                new_cluster_cnt[min_idx] += 1
            new_cluster.append(cluster_update)
```

각 SIFT feature들이 어떤 클러스터의 center에 가장 가까운지를 유클리디안 거리(L2 distance)를 통해 계산한 후, 가장 가

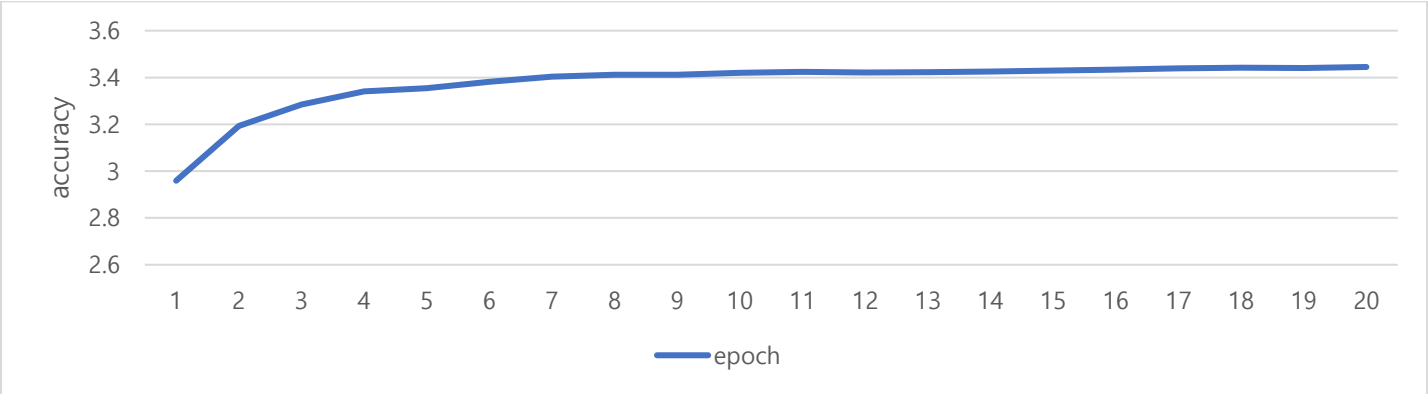
까운 클러스터로 분류한다. 이 정보는 new_cluster에 저장된다. 이후 클러스터의 center를 업데이트해야 하기 때문에 new_cluster_sum 에는 해당하는 SIFT feature를, new_cluster_cnt에는 1을 더해준다.

```
for i in range(0, N):
    for j in range(0, len(sifts[i])):
        k = new_cluster[i][j]
        # center update
        if new_cluster_cnt[k]==0:
            centers[k]=0
        else:
            centers[k] = new_cluster_sum[k] / new_cluster_cnt[k]
        # descriptor update
        d[i, k] -= np.log(np.linalg.norm(sifts[i][j] - centers[k], ord=2) + 1e-20) / len(sifts[i])
                # 거리가 0 일 경우를 위해 매우 작은 값을 더해줌
```

새롭게 업데이트된 cluster 의 center 를 계산한다. 위에서 각 new_cluster 마다의 sum, cnt 를 구해놓았기 때문에 new_cluster_sum[k] / new_cluster_cnt[k]로 평균을 계산해 center 를 업데이트한다. 만약 어떤 클러스터에는 아무 SIFT feature 가 분류되지 않았다면 center 는 0 으로 업데이트한다.
마지막으로 각 이미지의 descriptor 를 계산할 때, 각 SIFT feature 와 클러스터 center 의 유클리디안 거리에 로그 값을 취한 값을 빼준다. 유클리디안 거리는 각 SIFT feature 가 클러스터의 center 로부터 멀리 떨어져 있을수록 큰 값을 가지지만, 이 값들간의 차이를 줄이기 위해 로그 값을 취했다. 이 때 이미지마다 SIFT feature 의 개수 (n)의 개수가 다름에 따라 발생하는 차이를 줄이기 위해 len(sifts[i])=n 으로 나눠준 값을 빼서 descriptor 를 업데이트한다.

```
f = open('./A4_2017313008_' + str(epoch + 1) + '.des', 'wb')
N_ = np.array([N], dtype=np.int32)
D_ = np.array([D], dtype=np.int32)
f.write(N_.tobytes())
f.write(D_.tobytes())
f.write(d.tobytes())
f.close()
```

마지막으로 descriptor 파일의 포맷(binary)에 맞게 파일을 만든다.




```
(base) C:₩Users₩user₩PycharmProjects₩CV_A4>Eval.exe A4_2017313008.des
A4_2017313008.des -> L1: 3.4450 / L2: 2.9410
Your Accuracy = 3.445000
```

Epoch=20 수행 시, 최종 accuracy = 3.445