Computer Vision Assignment 3 Report

2014312692 ByeongKeonLee

1. K-means Algorithm

K-means 알고리즘은 모든 Center들을 기준으로 하여 내 Feature가 얼마나 떨어져 있는지 계산하고, 그 값들 중 최소값을 선택하여 그 Center에 종속되었다고 간주하는 알고리즘이다. K-means 알고리즘에서 처음 Center는 임의로 잡기가 가능하여, 현 과제에서는 각 이미지들의 Key값의 평균값을 잡았다.

K-means 알고리즘은 가장 먼저 임의로 Center들을 설정하는 것이다. Center의 개수는 우리가 나중에 뽑고자 하는 N by D 의 descriptor중에서 D의 값과 같다. 이 개수만큼 Center들을 임의로 설정한다. 하나의 image에는 key값이 대략 100개부터 1500개까지 다양하게 존재하는데, 이를 포괄한 모든 Key값 (760,512개)과 Center (D개, 최대 1,024개) 사이의 거리를 모두 계산한다. 하나의 Key값에는 128개의 데이터가 있고, Center도 마찬가지로 128개의 데이터가 있는데 이 사이에서는 유클리디안 거리를 사용하여 계산하였다. Key값 하나에서 Center까지의 거리의 종류는 D개의 종류가 존재하는데, 이 중 가장 작은 값을 채택하고, argmin을 사용하여 현재 계산된 Key값이 어느 Center에 속하는지 판단한다.

위의 과정을 모든 Key값에 대해서 수행하였다면, 이제 분류된 Key값들을 활용하여 새로운 Center의 좌표를 계산할 차례이다. 이 때는 한 Center를 기준으로 속하는 모든 Key값들에 대해서 평균값을 계산하고 이를 새로운 Center로 갱신한다.

위의 과정은 1 Epoch이다. 데이터의 보존을 위하여 해당 과제의 코드에서는 1 Epoch마다 descriptor를 출력한다. Descriptor를 계산할 때, 중심으로부터 떨어진 거리를 4제곱근을 활용하여 계산하였다. 중심으로부터 떨어진 거리가 멀면 그만큼 descriptor에 감산은 하지만, 선형성을 지니지는 않도록 하였다.

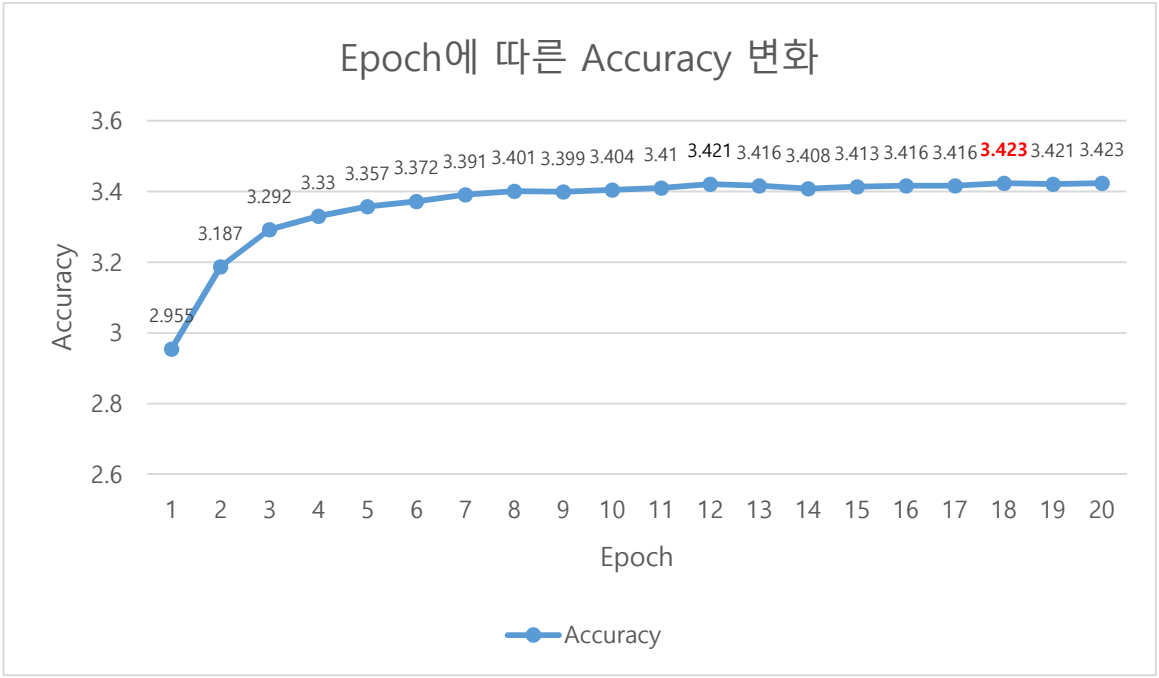
2. Code 설명

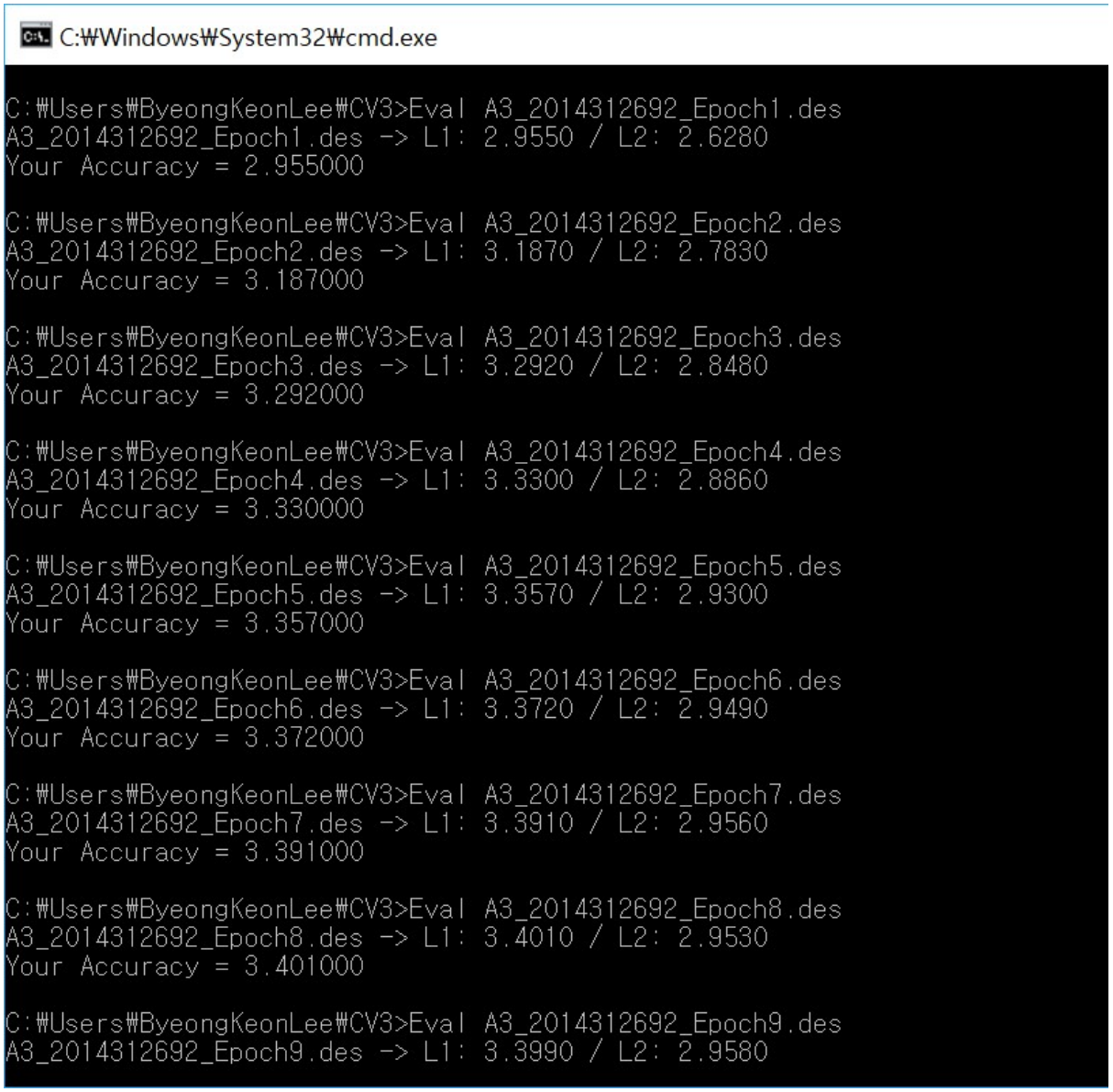
```
import cv2   #CV module import
import numpy as np   # numpy module import
# 760512
a=2 # input file zero number
N = 1000 # the number of images
D = 1000 # the number of descriptors per image
quotation = N//(D+1)+1   # Compaction constant of image descriptor
data = [] # key store structure
center = np.zeros((D,128)) # Each descriptors' center position
cluster = [] # Each key's   classified center
for i in range(N): # image's key loader
    if i==10 or i==100 :
```

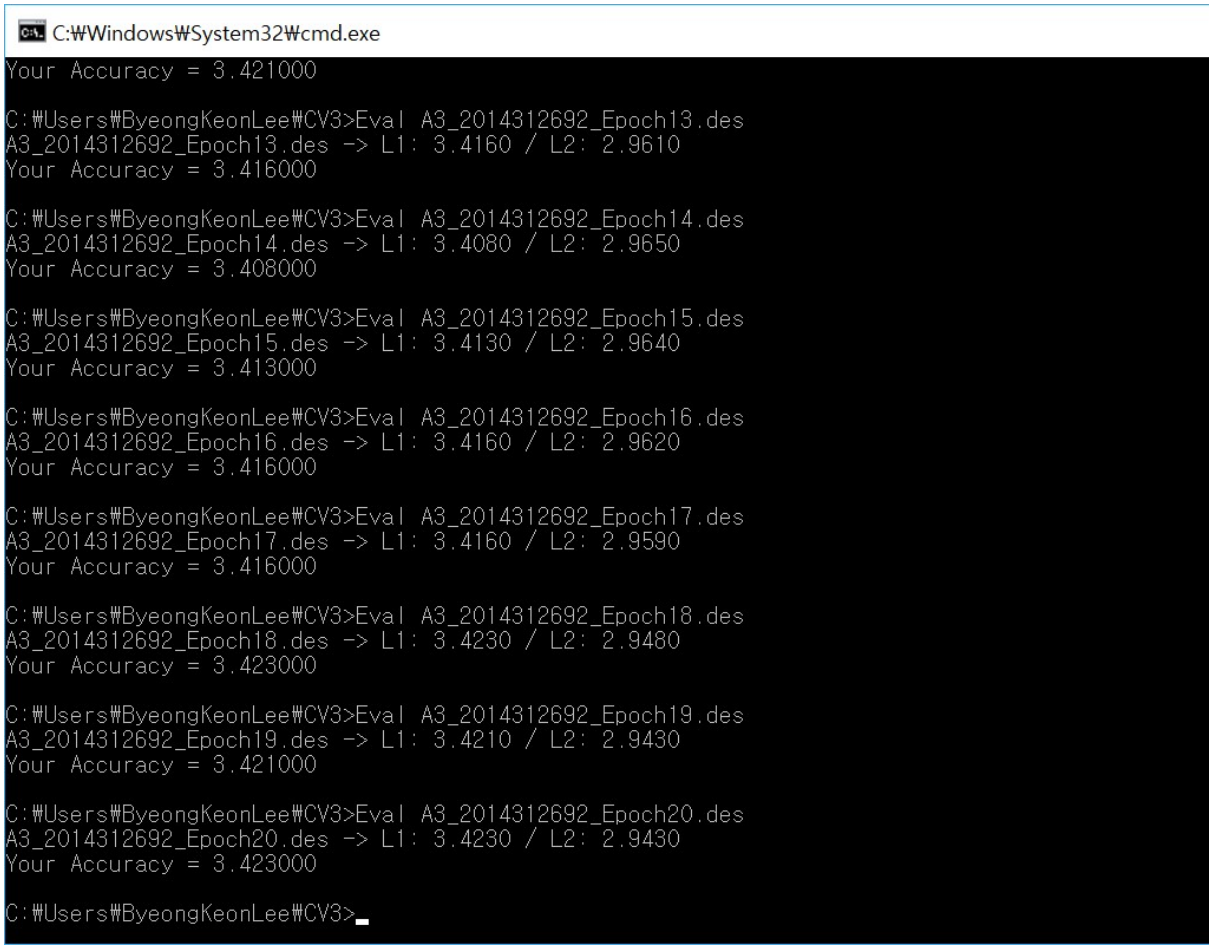
```
            a -=1
        f = open("./sift/sift100"+"0"*a+str(i),'rb') #open file
        s = f.read()
        tmp = np.reshape(np.frombuffer(s,dtype = np.uint8),(-1,128)) # read as int from buffer
        #if i<D:
        center[i//quotation,:] += np.average(tmp,axis=0) #set the center as image key average
        cluster.append(np.zeros((tmp.shape[0]),dtype=int))
        data.append(tmp)
        f.close()
#for i in range(N,D):
#       center[i, :] = data[i%N][0]
center = center/quotation # devide to make it average

#Kmeans
epoch=0
y = np.zeros((N, D)).astype(np.float32)
while epoch<100:
    for i in range(N):
            print(i)
            for j in range(data[i].shape[0]):
                cluster[i][j] = np.argmin(np.linalg.norm(data[i][j] - center,axis=1))   #find argmin which is the
index of minimum value of distance

        #center update
        center_update = np.zeros((D,128)) # center update key average store memory
        center_update_cnt = np.zeros((D,1)) # center update key count store memory
        for i in range(N):
        for j in range(data[i].shape[0]):
            center_update[cluster[i][j],:] += data[i][j] # sum the number of cluster key point
            center_update_cnt[cluster[i][j]]+=1  # count the number of cluster key point

        center = center_update / (center_update_cnt+1e-16) # center update as average of cluster
        epoch+=1

        x = np.array([N, D], dtype=np.int32) # prepare to print out N and D as file

        for i in range(N): # iterate the number of images
            for j in range(data[i].shape[0]):
        y[i, cluster[i][j]] -= np.power(np.linalg.norm(data[i][j]-center[cluster[i][j]]),0.25)/data[i].shape[0] # write
down the descriptor value as the distance between center and current key data

        #print(y, np.min(y[:,:],axis=0),np.max(y[:,:],axis=0))
        f = open("A3_2014312692_Epoch"+str(epoch)+".des", 'wb') # open the file
```

```
f.write(x.tobytes()) #write down x as byte
f.write(y.tobytes()) #write down y as byte
```

3. Epoch에 따른 Accuracy 변화




C:₩Windows₩System32₩cmd.exe
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch1.des
A3_2014312692_Epoch1.des -> L1: 2.9550 / L2: 2.6280
Your Accuracy = 2.955000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch2.des
A3_2014312692_Epoch2.des -> L1: 3.1870 / L2: 2.7830
Your Accuracy = 3.187000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch3.des
A3_2014312692_Epoch3.des -> L1: 3.2920 / L2: 2.8480
Your Accuracy = 3.292000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch4.des
A3_2014312692_Epoch4.des -> L1: 3.3300 / L2: 2.8860
Your Accuracy = 3.330000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch5.des
A3_2014312692_Epoch5.des -> L1: 3.3570 / L2: 2.9300
Your Accuracy = 3.357000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch6.des
A3_2014312692_Epoch6.des -> L1: 3.3720 / L2: 2.9490
Your Accuracy = 3.372000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch7.des
A3_2014312692_Epoch7.des -> L1: 3.3910 / L2: 2.9560
Your Accuracy = 3.391000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch8.des
A3_2014312692_Epoch8.des -> L1: 3.4010 / L2: 2.9530
Your Accuracy = 3.401000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch9.des
A3_2014312692_Epoch9.des -> L1: 3.3990 / L2: 2.9580


C:₩Windows₩System32₩cmd.exe
Your Accuracy = 3.421000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch13.des
A3_2014312692_Epoch13.des -> L1: 3.4160 / L2: 2.9610
Your Accuracy = 3.416000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch14.des
A3_2014312692_Epoch14.des -> L1: 3.4080 / L2: 2.9650
Your Accuracy = 3.408000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch15.des
A3_2014312692_Epoch15.des -> L1: 3.4130 / L2: 2.9640
Your Accuracy = 3.413000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch16.des
A3_2014312692_Epoch16.des -> L1: 3.4160 / L2: 2.9620
Your Accuracy = 3.416000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch17.des
A3_2014312692_Epoch17.des -> L1: 3.4160 / L2: 2.9590
Your Accuracy = 3.416000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch18.des
A3_2014312692_Epoch18.des -> L1: 3.4230 / L2: 2.9480
Your Accuracy = 3.423000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch19.des
A3_2014312692_Epoch19.des -> L1: 3.4210 / L2: 2.9430
Your Accuracy = 3.421000
C:₩Users₩ByeongKeonLee₩CV3>Eval A3_2014312692_Epoch20.des
A3_2014312692_Epoch20.des -> L1: 3.4230 / L2: 2.9430
Your Accuracy = 3.423000
C:₩Users₩ByeongKeonLee₩CV3>