ImageSearcher 보고서

지원자 : 김두훈

1. python 모듈 기본 요구사항

python(2.7.x)

scipy(0.16.1)

numpy(1.10.1)

nearpy(DataSeacher1.py)

PyMySQL (0.6.7)

openCV 2.4.x(manual Build)

2. Source Tree Overview

(ImageSearcher)

+- src

+- DataInserter

+- database

+- mysqlconn.py

+- \_\_init\_\_.py

+- improc

+- improc.py

+- \_\_init\_\_.py

+- test

+- test\_dbconn.py

+- \_\_init\_\_.py

+- util

+- util.py

+- \_\_init\_\_.py

+- DataSercher

+- database

+- mysqlconn.py

+- \_\_init\_\_.py

+- improc

+- improc.py

+- \_\_init\_\_.py

+- lshsearcher

+- lshsearcher.py

+- \_\_init\_\_.py

+- indexer

+- indexer.py

+- \_\_init\_\_.py

+- util

+- util.py

+- \_\_init\_\_.py

3. 중요 모듈 설명

* mysqlconn.py  
  pymysql을 연결하여 DB에 있는 Data를 Fetching하거나 Insert 하는데 사용한 모듈
* improc.py  
  image processing을 담당하는 모듈로써, Sift에 대한 처리를 한다.
* lshsearcher.py  
  nearpy의 LSH Encoding기능을 담당하기 위해 만든 모듈.
* util.py  
  각종 변환 기능들을 수행하는 모듈이다.
* indexer.py  
  직접 index 기능과 find 기능을 구현한 모듈로, lshsearcher의 기능과 비교를 위해 만들었다.
* datains.py  
  Feature Data를 DB에 삽입시키기 위한 모듈이다.
* DataSearcher1.py  
  lshsearcher의 기능을 이용하여 가장 넣은 데이터 중 가장 근접한 해쉬에 0에 가까운 근사치로 답을 준다.
* DataSearcher2.py  
  직접 구현한 구현한 것으로, DataSearcher는 가장 근접한 Hash만을 던져주어서 이미지에 대한 이름이 나오지 않는데, 이를 직접 구현하여 해결.

4. 구현 방법(요약)

* DataSearcher1.py  
  가장 단순하게, Database에 있는 Feature를 nearpy에 넣고(인덱싱 하고), 입력하는 이미지를 넣어 비교하는 방식이다.
* DataSearcher2.py  
  직접 구현한 Searcher로, 이는 다음의 순서로 요약 가능하다.  
  (1) 데이터 삽입  
   \* 입력 데이터셋의 중복 제거 및 소트  
   \* 부동소숫점형을 int형으로 변환  
   \* 딕셔너리를 만들어 매핑  
   \* 이름들을 인덱스 0부터 255까지에 맞게 삽입  
  (2) 데이터 찾기  
   \* 입력 데이터(찾으려고 입력 하는 데이터) 중복 제거  
   \* 부동소숫점형을 int형으로 변환  
   \* 입력 데이터의 번호 리스트들을 하나씩 기존에 만들어져 있는 딕셔너리에 찾아보며, 딕셔너리 안에 있는 이름 인덱스(다수중 하나)가 있으면 그 이름 인덱스 안에 값을 Vote한다  
   \* 최종적으로 작업이 완료 되면 리턴
* DataSearcher2의 idea  
  스케일 불면의 알고리즘인 sift의 특성 상, 만일 이 고유벡터에 맞는 모든 값들이나 일부분의 값들이 일치하게 된다면 어떻게 될까를 고민해보았으며, 그 Feature 내부 값 자체로도 충분히 다른 이미지들을 구분할 수 있을 가능성이 있다 판단되어 실험을 해보았다. sift에는 feature의 벡터에 중복값이 많았으며 Feature들의 고유값(Feature의 전체를 보고 이들의 엘리먼트 값들) 전체를 보고 중복이 많은 것을 알수 있었으며 이를 하나의 값만 남기고 모두 제거하여 딕셔너리 엘리먼트를 만드는 작업을 하였다. 이를 명확히 구조화 하여 만든 결과, 비슷한 벡터들을 가지는 이미지들끼리 모이는 것을 알 수 있었으며, 굳이 LSH를 쓰지 않아도 검색을 할 수 있다는 것을 이 구현을 통해서 알게 되었다.

5. 보고 및 평가

\* 영상처리 작업 부분 보고

영상 이미지를 LSH로 비교하기 위해선 우선은 이미지에서 Feature를 생성해야 하는데, 이 Feature에 대한 생성을 할 경우에 간혹가다가 원래 생성하려고 하는 Feature의 갯수보다 크거나, 아니면 이미지 자체가 Feature를 많이 발견하지 못하기 때문에 배열을 늘이거나 혹은 줄이는 작업을 해야한다. 이를 만회하기 위해서 이미 Inserter 및 Searcher에 이미 이를 구현하였다.

\* 구현 작업 보고 및 총평

sift와 LSH를 사용하여 이미지 검색을 구현하는 것을 처음 구현해보았다. 전체적으로는 어렵다고 생각한다.

해당 영상을 Sift로 변환하여 만든 Feature 정보는 항상 Scale invarient하다. 따라서 LSH를 썼을때 어떠한 벡터의 일부분은 비교하는 이미지와 거의 동일한 Document일 것이고, 따라서 hash값이 일치 할 가능성이 높다(Hash의 Collision 현상을 최대한 활용). 또한, sift의 벡터의 각도는 0부터 255 미만(영상에서 생성되는 벡터의 최소 및 최대값)이며 따라서 배열이 정수로 나오게 된다. 따라서 sift의 scale invarient한 점 자체가 signiture가 되기 때문에(도큐먼트에 있는 벡터 자체가 스케일에 ‘거의’ 불변하는 고유 벡터이기 때문에) 이에 대한 아이디어를 얻어 약 3시간정도를 DataSearcher2를 구현하여 시험해보았다.

\* 성능

구현된 구현물을 비교해본 결과, DataSearcher1의 LSH 해싱은 해시값 매칭으로 인덱싱 시간이 평균 30초 정도 걸리며 이를 커밋하는데 걸리는 시간까지 포함하면 상당히 오래 걸리는 수준이다. 하지만, 찾는 속력은 평균 0.05초 미만이었다.

또한, DataSearcher2는 인덱싱에서 2분이 넘는 시간이 걸리며, 인덱싱을 찾는 속력 또한 평균 2초정도 였다.

DataSearcher2는 개선의 여지가 있으며, 많은 작업을 통해 간단하게 나마 성능을 0.5초 이내로 개선이 가능할 것으로 보인다.

6. 가장 어려웠던 점  
우선은 LSH 및 모르는 부분이 많았다. 문제에서는 ElasticSearch를 이용하라고 하였지만, 조금 더 로컬하게 성능을 올리고자 하는 마음에서 ElasticSearch를 이용하지 않았으며, python용 LSH 라이브러리들이 상당히 불편한 예제를 가지고 있었으며, 구조적으로 이를 해결하기 위해 개조하여 lshash 같은 라이브러리들을 개조하여 사용하고, 테스팅까지 하였지만 사실상 원하는 기대치에 못미쳐서 DataSearcher1은 nearpy를 사용하였다.  
구현을 위한 검색과 연구, 구체적인 테스팅 및 트러블 슈팅들에 쏟은 시간이 구현에 약 3배 가량 되었던 것으로 추산된다.  
하지만, 상당히 좋은 경험이었으며, 모르는 알고리즘에 대한 자세한 이해와 함께 이를 어디에 이용하는지, 그리고 어떻게 사용해하여 성능상의 이득을 볼지, 또한 이러한 구조들에 대한 지원자 김두훈만의 생각과 이에 대한 아이디어를 구체적으로 구성하여 실행에 옮겼던 것은 굉장히 고무적이었다.  
재미있는 구현 과제를 주신 버즈니 기술팀에게 감사의 말씀을 전합니다.

지원자 김두훈 드림.