Rasterized Image Databases for Image Compression

Zoya Bylinskii*, Maria Shugrina*, Andrew Spielberg*, Wei Zhao*

CSAIL

bigdata@CSAIL
MIT BIG DATA INITIATIVE

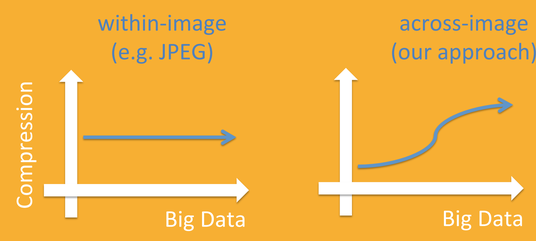
6.830
Database Systems
Final Project

MOTIVATION

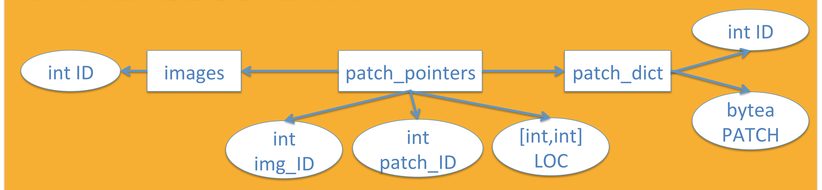
- More than 1.8 billion images uploaded to the internet every day
- Redundancy in large image collections can lead to more efficient storage

PATCH-BASED IMAGE CODING

- Lossy compression
- Compression savings INCREASE as the database size INCREASES
- Rasterization is a simple operation – easily deconstruct and reconstruct images using patches



DATABASE SCHEMA



METHOD OVERVIEW

SCHEMA:

```
images(id int PRIMARY KEY);

patch_dict(id int PRIMARY KEY, patch bytea);

patch_pointers(img_id int REFERENCES images(id),
               patch_id int REFERENCES patches(id),
               x int, y int);
```

ALGORITHM:

Algorithm 1 Insert Image I into database

1: $Patches \leftarrow$ `Patchify`(I, n)
2: **for** P_j in $Patches$ **do**
3: $SimPat \leftarrow$`FindLikelySimilarPatches`$(P_j, patch\_dict)$
4: $P_{NN} \leftarrow argmin_{P_i \in SimPat}\{S(P_i, P_j)\}$
5: **if** $S(P_{NN}, P_j) > T$ **then**
6: `insert` P_j into `patches`

Algorithm 2 Modification of alg. 1, with ANN

1: $Patches \leftarrow$ `Patchify`(I, n)
2: **for** P_j in $Patches$ **do**
3: $SimPat \leftarrow$`FindLikelySimilarPatches`$(P_j, patch\_dict)$
4: $P_{ANN} \leftarrow M(P_j)$
5: **if** $S(P_{ANN}, P_j) > T$ **then**
6: `insert` P_j into `patches`

Algorithm 3 Optimization of alg. 2, with ANN

1: $Patches \leftarrow$ `Patchify`(I, n)
2: $UniquePatches \leftarrow$ `FindUniquePatches`(I)
3: $PatchesToStoreInDB \leftarrow []$
4: **for** P_j in $UniquePatches$ **do**
5: $SimPat \leftarrow$`FindLikelySimilarPatches`$(P_j, patch\_dict)$
6: $P_{ANN} \leftarrow M(P_j)$
7: **if** $S(P_{ANN}, P_j) > T$ **then**
8: $PatchesToStoreInDB$.Add(P_j)
9: `BatchInsert`$(PatchesToStoreInDB)$

Distance per color channel in CIE (LUV) color space:

$$S(P_i, P_j, u) = \frac{||P_i(u) - P_j(u)||^2}{n^2}$$

Thresholds T1, T2, T3 per channel.

Image size mxm, pach size nxn.

KEY OPTIMIZATIONS:

- Minimize database queries (batch query per image)
- Near-neighbor search
- Patch BufferPool for patches (LRU)
- Minimize re-computation of image vectors
- Exploit image self-similarity

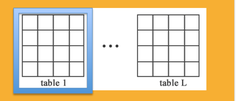
IMPLEMENTATION:

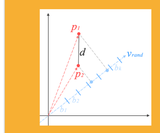
Posgresql database with Java API.
Image loading, patching, searching, hashing, Near-Neighbor search, reconstruction implemented in Java.

https://github.com/shumash/db_project

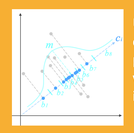
NEAR NEIGHBOR SEARCH

LSH: AND and OR hashes for high precision/recall.
Can we find one AND hash for reasonable P1, low P2?





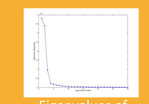
Random-projection hash (we use AND of 10 in 32-bit int): poor P1, P2



Using PCA directions as projection vectors to improve P1, P2

$$P[b(P_i) = b(P_j) | S(P_i, P_j < T)] > P_1$$
$$P[b(P_i) = b(P_j) | S(P_i, P_j) > cT] < P_2$$

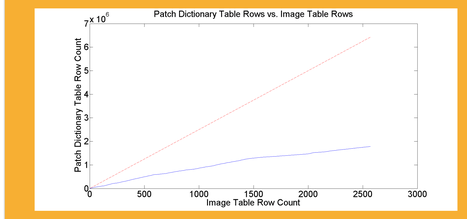
PCA-BASED HASHING:
Pick projection vectors to be directions of maximal variance in a subset of images.



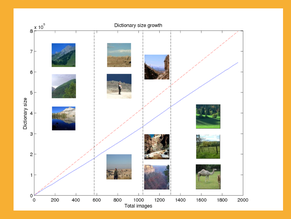
Eigenvalues of patch sample.

ADDITIONAL INSIGHT:
Natural images contain many nearly uniform patches, which are easy to index and search:
`uniform_patch_dict`

GENERALIZABILITY

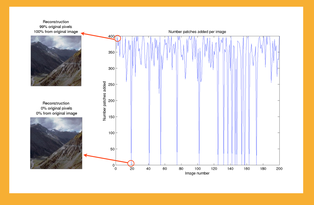


Generalizability across image types:



APPLICATIONS

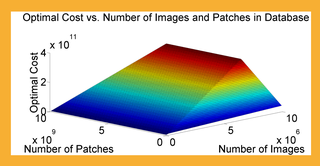
- duplicate detection
- similar image retrieval



EXTENSIONS

- context-aware patch sampling
- task-aware threshold selection
- patch hierarchies

COST CALCULATIONS



$$\underset{patch\_dict, M}{\text{minimize}} \quad c(k, d, m, n)$$
$$\text{subject to} \quad S(P_j, M(P_j)) \leq T, \; j = 1, \ldots, k\left(\frac{m}{n}\right)^2$$

Cost without patch-based scheme: $c'(k, m) = 3km^2$

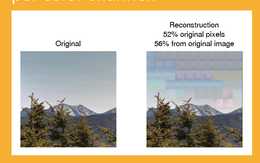
Cost with patch-based scheme: $c(k, d, m, n) = 8k\left(\frac{m}{n}\right)^2 + 3dn^2$

Constraint for cost savings: $d < \frac{m^2(3n^2 - 8)k}{3n^4}$

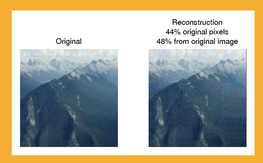
PARAMETER SELECTION

A number of factors, including the patch size, distance function, and threshold affect the costs of the patch-based compression scheme.

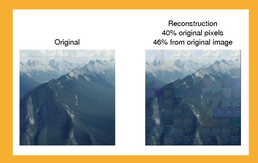
The importance of the right color space and a threshold per color channel:



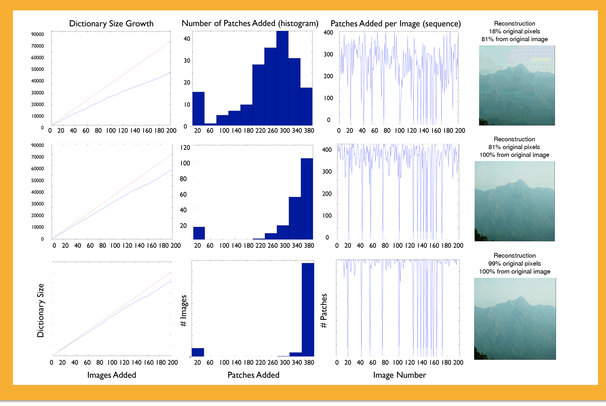
The importance of the right patch size:



The importance of the right distance function and threshold:



Weighting storage savings and reconstruction speed against compression quality:



The importance of the right patch sampling technique: