Rasterized Image Databases with LSH for Compression, Search and Duplicate Detection

Zoya Bylinskii
MIT CSAIL

Maria Shugrina
MIT CSAIL

Andrew Spielberg
MIT CSAIL

Wei Zhao
MIT CSAIL

ABSTRACT

TODO.

1. INTRODUCTION

Large collections of images are ubiquitous in the modern digital world. According to one 2014 Internet Trends report, more than 1.8 billion images are uploaded to the internet every day [3]. Our work is inspired by the intuition that there must be a lot of redundancy in large image collections, and that this redundancy could be exploited for more efficient storage and for applications such as duplicate detection.

We focus on image redundancy on the patch level, assuming that large collections of images must have many patches which are nearly the same. Our goal is to store a set of images as a database of similar patches, where similar patches may be shared between images, such that we minimize the storage space while maintaining certain *quality* of reconstructed images. In effect, this results in lossy compression. More concretely, we aim to choose patch similarity criterion, and search and reconstruction algorithms such that:

- the database size is smaller than if full images were stored
- the images can be reconstructed from the database in real time
- the reconstructed images fulfill certain quality requirements (See Sec. 4)

These conflicting introduce a number of tradeoffs, such as size of the database versus image quality. The goal of this paper is as much to produce a working system as to build up the analytical foundations that allow making these tradeoffs.

In Section 3, we explain our method of "patchifying" images, storing them in a database, and reconstructing stored images. We also discuss image similarity, hashing schemes and indices in this core section. In Section 4, we provide analytical groundwork for selecting optimal image patch sizes, similarity threshold, and the quality metrics appropriate for evaluation. Finally, in Section 5 we evaluate our system on real data, using analytical tools from the previous section. In addition, in Section 6, we briefly touch on applications that come naturally from a patch database, including similar image retrieval and duplicate detection.

2. RELATED WORK

Image databases, in particular rasterized.
Image compression, in particular JPEGs.
Image similarity functions.
Image quality functions.
Hashing, in particular [1].

3. METHOD

To limit the complexity, we assume each image to be a square of m^2 pixels, and for all patches to be squares of the same size. We formally describe our method in Sec. 3.1. To summarize, we first segment each image into patches and store them in the `patches` database. Only patches sufficiently different from all the patches in `patches` are stored (See Sec. 3.2). Instead of storing each image explicitly, we approximate it by storing pointers to the patches approximating its original patches. In Sec. 3.3, we describe the search algorithm used to quickly retrieve similar patches, and in Sec. 3.4 we detail the indices that make image reconstruction faster. Finally, in Sec. 3.5 we provide implementation details.

3.1 Overview

Our method is governed by the following parameters:

- n - width and height of all patches[1]
- S - similarity function from two $n \times n$ images to $\mathbb{R}$
- T - similarity threshold

The $n \times n$ patches are stored as byte data in the `patches` table. To allow image reconstruction, we also store $(m/n)^2$ patch pointers for each image in the `patch_pointers` table. The full schema looks as follows:

```
patches(id int PRIMARY KEY,
        patch bytea);

images(imgid int PRIMARY KEY);
```

[1] We assume that $m \mod n$ is 0.

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

where `patch_pointers.x` and `patch_pointers.y` refer to the left top corner location of each patch in the image.

Given these tables, inserting an image into the database proceeds as follows:

Algorithm 1 Insert Image I into database

```
$Patches \leftarrow$ CutIntoPatches(I, patch_size=$n$)
for $P$ in $Patches$ do
    $SimPat \leftarrow$FindLikelySimilarPatches($P$)
    $P_{closest} \leftarrow argmin\{S(P, P_i)\}$
    if then$S(P, P_i) > T$
        insert $P$ into patches
```

Sections 3.2 and 3.3 detail similarity measure and finding patches that are likely to be similar.

In order to reconstruct an image from that patches, we run the following procedure:

3.2 Patch Similarity

There are many image similarity metrics that have been developed for images (See [6] for a good survey.), and our method is applicable to any metric that involves Euclidean distance over image features, its stacked color channel pixel values being the simplest case.

For the purpose of this project, we choose to use squared Euclidean distance over (CIE)LUV color space. Given two $n \times n$ patches P_1 and P_2, we evaluate similarity S per color channel i as follows:

$$S(P_1, P_2, i) = \frac{||P_1(i) - P_2(i)||^2}{n^2}$$

where $|| \cdot ||$ denotes standard Euclidean norm. We normalize by the dimensionality of the space to allow us to keep the similarity threshold independent of the patch size. See section 4.3 for more details.

The use of Euclidean distance for patch similarity allows us to use Locality Sensitive Hashing to retrieve patches that are likely to be similar, as detailed in the next section.

3.3 LSH for Patches

3.4 Image Reconstruction

In order to reconstruct images quickly, we

3.5 Implementation

We used *postgresql* to construct our database, and used Java API to talk to the database from a custom executable. Locality sensitive hashing, image segmentation and reconstruction were all implemented in Java, and used to construct a hash table on patches in *postbresql*.

Our code is available at:

`https://github.com/shumash/db_project`

4. ANALYSIS

A number of parameters can be tweaked to change the patch matching and storage, and different choices may be appropriate for different applications and performance requirements (both quantitative and qualitative). These parameters include the size of the input images, the size and shape of the patches extracted, the extraction strategy (non-overlapping vs overlapping patches), the sampling strategy used to seed the dictionary, the similarity metric and thresholds used to compare patches, as well as the parameters required for indexing and retrieving patch matches (approximate nearest neighbors). Here we discuss some of the parameter choices made and the experiments that lead up to these choices. Other possible choices are discussed in Sec.7.1.

Our quantitative performance metrics involve examining how the patch dictionary size grows with the addition of new images to the database (the growth function and rate) and the compression ratio per image (viewed as a distribution over compression ratios and summarized as the average compression ratio). Qualitative evaluations involve determining whether a human can spot compression artifacts and how salient they are in the images. The authors of this paper manually examined images reconstructed from the dictionary patches. A crowdsourced evaluation strategy involving Amazon's Mechanical Turk may be appropriate for larger-scale studies, but was beyond the scope of this paper.

There will always be a trade-off between compression benefits (storage - patch dictionary size, speed - image reconstruction time) and reconstruction quality. For many computer vision tasks including scene recognition (and thus retrieval), imperfect reconstructions with artifacts may not be a problem as long as the overall scene structure does not change. For instance, [4] has shown that with images of pixel dimension 32x32, humans are already able to achieve over 80% scene and object recognition rate. See fig.1 for a demonstration of an image that has serious reconstruction artifacts, but when down sampled (to a thumbnail), they become insignificant, and thus do not necessarily impair visual recognition.

4.1 Patch Size

TODO(Zoya,Andrew): tradeoffs for patch size

Assume for now that we choose to store p patches in our auxiliary table. In practice, we choose p to be a function p: $Function \rightarrow \mathbb{N}$ which maps from our similarity metric to a number of patches to store. Assume also that each patch is square and composed of n^2 pixels, where n is a user defined parameter. We further assume that each pixel requires 8 bytes to store and that each pointer is 8 bytes (a standard integer for a 64-bit system). Under this "image only" scheme, in the case where we have i images, the cost c_i to store all the images in our database is:

$$c_i(i, m) = 8im^2 \quad (1)$$

In the case where we store pointers to patches, we have two tables: one table to store pointers to image patch exemplars, and a second table to store the exemplar data themselves. Under this "patch pointer"scheme, in the case where we have i images and p patches, the cost c_p to store all the images in our database is:

$$c_p(i, p, m, n) = 8i(\frac{m}{n})^2 + 8pn^2 \quad (2)$$

.

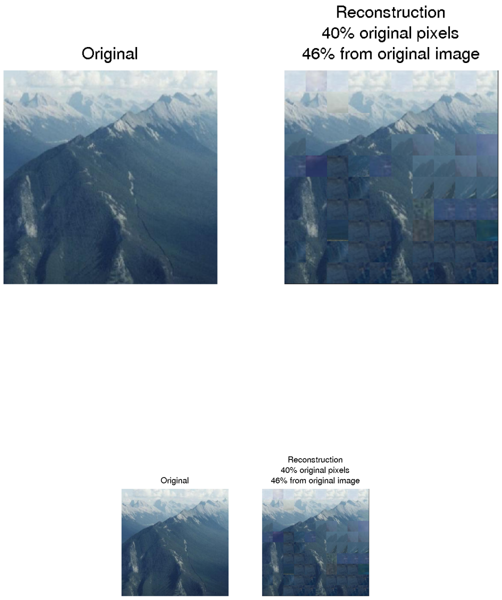

Figure 1: For demonstration purposes only, we choose a large patch size and low similarity threshold. Under these parameters, the original image is reconstructed to take up only 40% of its original size (in pixels). The 60% of the patches that have been replaced come either from the same image (46% of them), or from other images (the remaining 64%). Notice that when the size of the image and its reconstruction are halved, the artifacts already become visually insignificant, and would not impair a scene recognition or search task.

The first term is the cost of storing the pointer data, while the second term is the cost of storing the patch exemplars themselves.

4.2 Sampling strategies

A patch dictionary can be built up incrementally, adding new patches as new images are added to the database. A potential problem with this approach is that image reconstruction quality will tend to decrease with the order in which images are added, such that images added to the database earlier will tend to have more patches that correspond to them (see Fig.2 for an example). A strategy with a more even distribution of reconstruction quality over images involves starting with a batch of images, and seeding the dictionary by randomly sampling patches from a set of images from the batch. This is the strategy we employ.

4.3 Similarity Threshold

Many image (more specifically, patch) similarity functions are possible, each with its own distinct set of parameters that can be tweaked for the required application. Because we are dealing with patches of a size specifically chosen to increase within-patch homogeneity, we do not consider cases of patches containing objects (the most we expect is an object boundary or simple texture), and thus do not need to consider complex image similarity functions (like SIFT matching, spatial relationship-preserving functions, etc.). We can constrain ourselves to color similarity, and split a patch P_i into 3 color channels: $P_i(1), P_i(2), P_i(3)$.

Then we consider two patches P_i and P_j similar when, given $n \times n$ patches, all of the following are true:

$$\frac{1}{n^2}||P_i(1) - P_j(1)||^2 < T_1$$

$$\frac{1}{n^2}||P_i(2) - P_j(2)||^2 < T_2$$

$$\frac{1}{n^2}||P_i(3) - P_j(3)||^2 < T_3$$

The $\frac{1}{n^2}$ term allows us to normalize for patch size, so that the threshold values chosen becomes independent of patch size. Here we constrain the average similarity value of all the pixels to fit a threshold, whereas it is possible to have alternative constraints (whereby the maximal pixel difference, or the variance of the pixel differences, or some other quantity does not exceed a threshold).

Note additionally that if instead, we fix a single threshold for the sum of the Euclidean differences in the 3 color channels:

$$\frac{1}{n^2}[||P_i(1)-P_j(1)||^2+||P_i(2)-P_j(2)||^2+||P_i(3)-P_j(3)||^2] < T$$

then the similarity in one color channel may compensate for the difference in another, producing skewed results (see fig.3).

Multiple color channels $P_i(1), P_i(2), P_i(3)$ are possible, but we choose to work in the (CIE)LUV color space, which is known to be more perceptually uniform than the standard RGB color space [2]. Additionally, our formulation makes it possible to impose separate similarity thresholds on each of the color channels (T_1, T_2, T_3). However, for simplicity, we set $T_1 = T_2 = T_3$.

5. PERFORMANCE

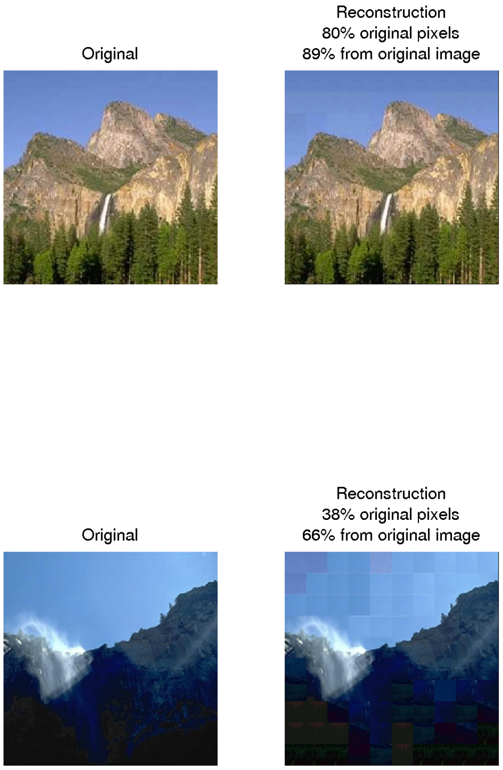

Figure 2: Example of a biased patch dictionary construction strategy, leading to non-uniformity in image reconstruction quality. Images added to the database earlier (top row) are better reconstructed (due to more patch samples in the database) than images added later (bottom row), constrained to be constructed out of patches added initially. The sky pixels in the image added later are borrowed from sky pixels of other images (44% of the pixels in this image come from other images, compared to only 11% in the image on the first row). Note: here we use a very low patch similarity threshold and large patch size for demonstration purposes only, to emphasize the artifacts created.

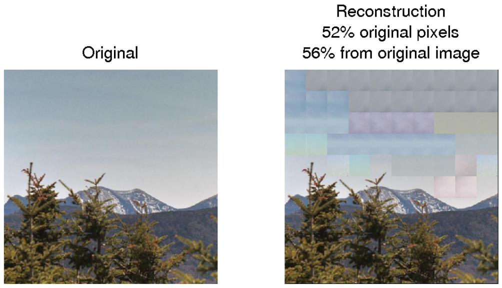

Figure 3: This is what happens when we do not separately constrain each of the color channels to match. We have patches that match in terms of general hue (average of the color channels), but are the wrong color and produce visible visual artifacts.

In this section we evaluate the performance of the implementation of our method on a real collection of X images of Y size.

5.1 Quantitative

5.2 Qualitative

6. APPLICATIONS

One of the appeals of this approximate patch-based approach is that it naturally lends itself to applications. In this section we describe methods to use our database for two applications - duplicate detection (Sec. 6.1) and similar image retrieval (Sec. 6.2).

6.1 Duplicate Detection

6.2 Similar Image Retrieval

7. CONCLUSION

7.1 Future Extensions

In this paper we considered a simple implementation of a patch-based image database compression scheme, where the images and patches were square and of fixed sizes. Patches were sampled in a regular, non-overlapping grid from each image. Alternative approaches include more flexible, context-aware, patch-sampling techniques. For instance, the patch granularity for sampling large homogenous sky and field regions may be different from the one used for sampling highly-textured regions like objects and structures (trees, buildings, people, etc.). Similarly, patches that do not cross object boundaries are likely to lead to less artifacts in future reconstructions. For this, approaches like Selective Search [5] that localize image regions likely to contain objects, may prove promising for sampling patches.

8. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, Jan. 2008.

[2] H. Kekre and V. K. Banura. Performance comparison of image retrieval using kfcg with assorted pixel window sizes in rgb and luv color spaces 1. 2012.

[3] M. Meeker. Internet trends 2014 - code conference, 2014.

[4] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large database for non-parametric object and scene recognition. *IEEE PAMI*, 30(11):1958–1970, November 2008.

[5] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[6] M. Yasmin, M. Sharif, and S. Mohsin. Use of low level features for content based image retrieval: Survey. *Research Journal of Recent Sciences*, 2277:2502, 2013.