AN ENSEMBLE DIVERSITY APPROACH TO SUPERVISED BINARY HASHING

Miguel Á. Carreira-Perpiñán and **Ramin Raziperchikolaei**, EECS, UC Merced

1 Abstract

Binary hashing is a well-known approach for fast approximate nearest-neighbor search in information retrieval. Much work has focused on affinity-based objective functions. These objective functions encode neighborhood information between data points. They ensure that the hash functions differ from each other through constraints or penalty terms that encourage codes to be orthogonal or dissimilar across bits, but this couples the binary variables and complicates the already difficult optimization. We propose a much simpler approach: we train each hash function (or bit) independently from each other, but introduce diversity among them using techniques from classifier ensembles. Surprisingly, we find that not only is this faster and trivially parallelizable, but it also improves over the more complex, coupled objective function.

Work supported by NSF award IIS–1423515

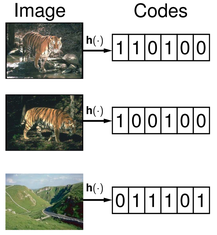
2 Binary hash functions for fast image retrieval

In K nearest neighbors problem, there are N training points in D-dimensional space (usually $D > 100$) $\mathbf{x}_i \in \mathbb{R}^D$, $i = 1, \dots, N$. The goal is to find the K nearest neighbors of a query point $\mathbf{x}_q \in \mathbb{R}^D$. Exact search in the original space is $\mathcal{O}(ND)$ in time and space.

A binary hash function $\mathbf{h}$ takes as input a high-dimensional vector $\mathbf{x} \in \mathbb{R}^D$ and maps it to an b-bit vector $\mathbf{z} = \mathbf{h}(\mathbf{x}) \in \{0,1\}^b$.

The main goal is preserving the neighborhood, i.e., assign (dis)similar codes to (dis)similar patterns.

In supervised hashing, we try to preserve tshe semantic similarity between the images (e.g. images from different view points are similar, while they are far in the Euclidean space).



Finding K nearest neighbors in Hamming space needs $\mathcal{O}(Nb)$ in time and space. Distances can be computed efficiently using hardware operations.

$N = 10^9$, $D = 500$ and $b = 64$

Search in	Space	Time
Original space	2 TB	1 hour
Hamming space	8 GB	10 seconds

3 Affinity-based objective functions

Most hashing papers try to minimize an affinity-based objective function, which directly tries to preserve the original similarities in the binary space:

$$\min \mathcal{L}(\mathbf{h}) = \sum_{n,m=1}^{N} L(\mathbf{h}(\mathbf{x}_n), \mathbf{h}(\mathbf{x}_m);\ y_{nm})$$

where $\mathbf{x}_i \in \mathbb{R}^D$ is the ith input data, $\mathbf{h}$ is the parameters of the hash function, $L(\cdot)$ is a loss function that compares the codes for two images with the ground-truth value y_{nm} that measures the affinity in the original space between the two images $\mathbf{x}_n$ and $\mathbf{x}_m$. Many such loss functions $L(\mathbf{z}_n, \mathbf{z}_m;\ y_{nm})$ exist, e.g.:

KSH: $(\mathbf{z}_n^T \mathbf{z}_m - b y_{nm})^2$ Laplacian: $(y_{nm} \|\mathbf{z}_n - \mathbf{z}_m\|^2)$ BRE: $(\frac{1}{b}\|\mathbf{z}_n - \mathbf{z}_m\|^2 - y_{nm})^2$

4 Optimization-based approaches

Optimizing $\mathcal{L}(\mathbf{h})$ is difficult because $\mathbf{h}$ is discrete. Many optimization algorithms have been considered in the binary hashing literature:

- Relaxation (Liu et al., 2012): relax the step function or binary codes (ignore the binary nature of the problem), optimize the objective continuously, and truncate the results.
- Two-step methods (Lin et.al., 2013, 2014): in the first step, define the objective over binary codes, optimize it approximately and learn the codes, and in the second step, fit the hash function given the codes.
- MAC (Raziperchikolaei and Carreira-Perpiñán, NIPS 2016): this achieves the lowest objective value by respecting the binary nature of the problem and optimizing the binary codes and the hash function jointly.

Limitations of optimization-based methods:

- The hash function outputs binary values, hence the problem is nonconvex and nonsmooth. The underlying problem of finding the binary codes for the points is an NP-complete optimization over Nb variables.
- Most methods do not scale beyond a few thousand training points.
- The b single-bit hash functions are coupled (to avoid trivial solutions where all codes are the same).
- In the end, there is little practical difference between the different objective functions and optimization algorithms proposed.

Is optimizing all the functions jointly crucial anyway? In fact, it isn't.

5 An ensemble diversity approach

Rather than coupling the b hash functions into a single objective function, we propose to train each hash function independently from each other.

We minimize the following objective over the single-bit hash function $h\colon \mathbb{R}^D \to \{-1, +1\}$, b times independently:

$$\min \mathcal{L}(h) = \sum_{n,m=1}^{N} L(h(\mathbf{x}_n), h(\mathbf{x}_m);\ y_{nm}).$$

To minimize this objective, we could use any of the previous optimization-based methods. Here, we first define the objective over the codes and then fit the hash function. We use the Laplacian loss in experiments and that is why we call our method Independent Laplacian Hashing (ILH).

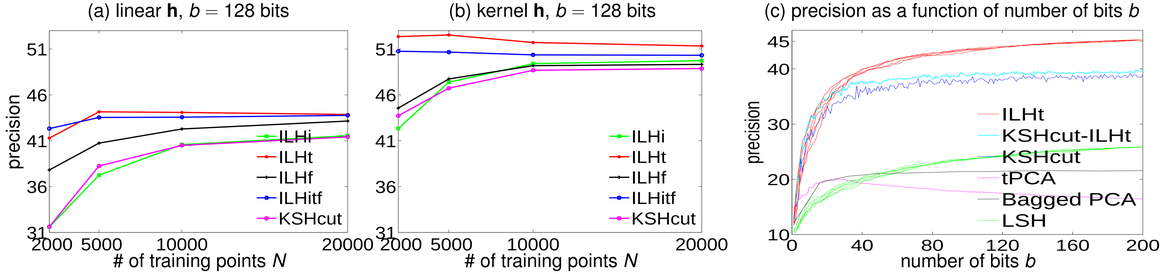
To get good retrieval results, we need the single-bit hash functions to be different from each other. We use the ensemble learning techniques to make the hash functions different from each other:

- Different initializations (ILHi): Each hash function is initialized randomly, which results in different local optima for different hash functions.
- Different training sets (ILHt): Each hash function uses a training set of N points that is different from the other hash functions.
- Different feature subsets (ILHf): Each hash function is trained on a random subset of $1 \le d \le D$ features.

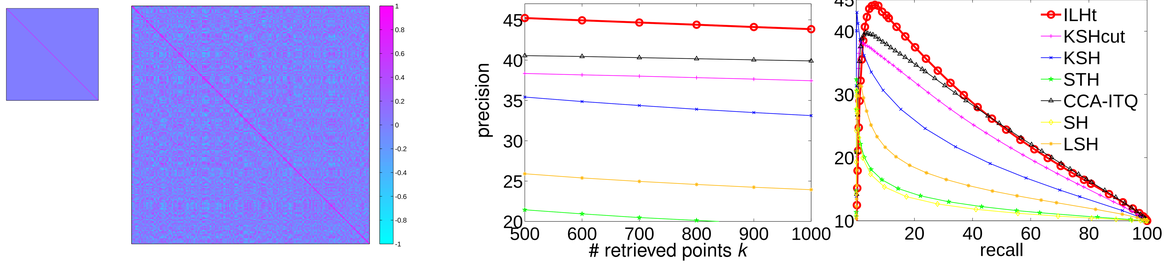
6 Advantages of our approach

- In terms of retrieval performance (precision/recall), our ensemble diversity approach is better or comparable to optimization-based methods.
- Much simpler optimization: we deal with b independent problems each over N binary codes rather than 1 problem with Nb binary codes.
- Hence, faster training and better accuracy, because we deal with optimization problems of a smaller size.
- Training the b functions can be parallelized: this helps to learn a large number of single-bit functions very fast. One can then use pruning to select a small subset of them that has comparable retrieval performance (Raziperchikolaei and Carreira-Perpiñán, ICDM 2016).
- To get the solution for $b+1$ bits we just need to take a solution with b bits and add one more bit: helpful for model selection.
- For ILHf, both the training and test time are lower than if using all D features for each hash function. The test runtime is d/D smaller.

7 Experiments



(a) and (b). Comparing different diversity mechanisms and their combination (ILHitf) with each other and with the state-of-the-art method KSHcut (Lin et.al, 2014), using linear and kernel hash functions. ILHt (using different training sets) clearly outperforms the other methods.
(c). Precision as a function of number of bits. We compare ILHt with state-of-the-art method KSHcut which uses optimization-based approach, and also baseline methods LSH (truncated random projection) and tPCA (truncated PCA). ILHt outperforms other methods and its precision increases nearly monotonically. Optimizing KSHcut using the ILHt codes as initialization (KSHcut-ILHt) remains far from ILHt.



(d) and (e). Are the codes of ILHt orthogonal? Define $b \times b$ matrix $\mathbf{C_Z} = \frac{1}{N}\mathbf{Z}^T\mathbf{Z}$ with the entries $(\mathbf{z}_n^T\mathbf{z}_m)/N \in [-1, 1]$. The code matrix is orthogonal if we have: $\mathbf{C_Z} = \mathbf{I}$. Looking at the codes of ILHt (part (e)) It does not seem necessary to enforce orthogonality to achieve good binary codes.
(f). We compare ILHt with the other hashing methods by reporting precision and precision-recall curves. ILHt beats other hashing methods.