Learning independent, diverse binary hash functions: pruning and locality

Ramin Raziperchikolaei and **Miguel Á. Carreira-Perpiñán**

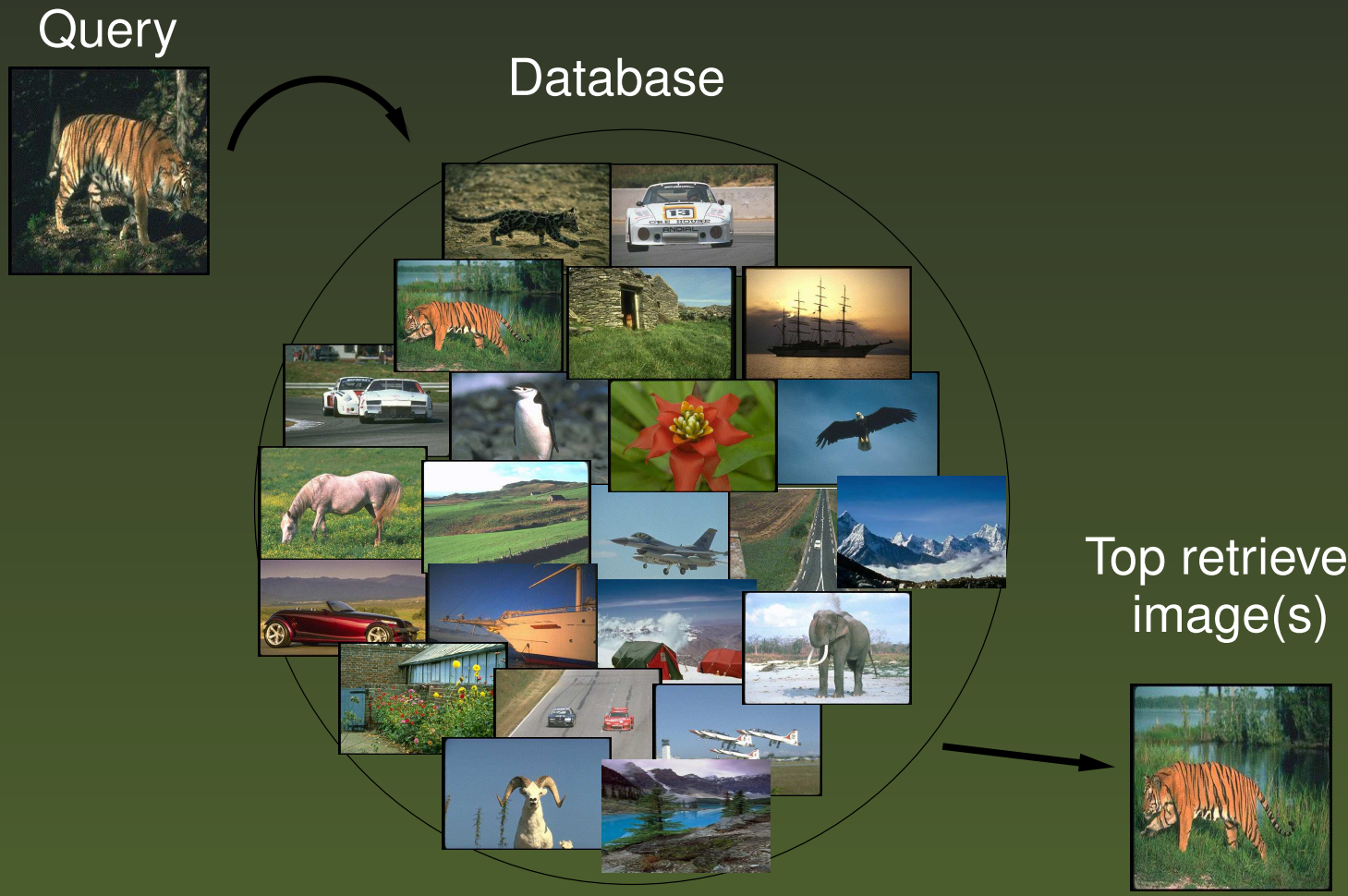
Electrical Engineering and Computer Science

University of California, Merced

http://eecs.ucmerced.edu

Large scale image retrieval

Searching a large database for images that are closest to a query.
A nearest neighbours problem on N vectors in $\mathbb{R}^D$ with large N and D.

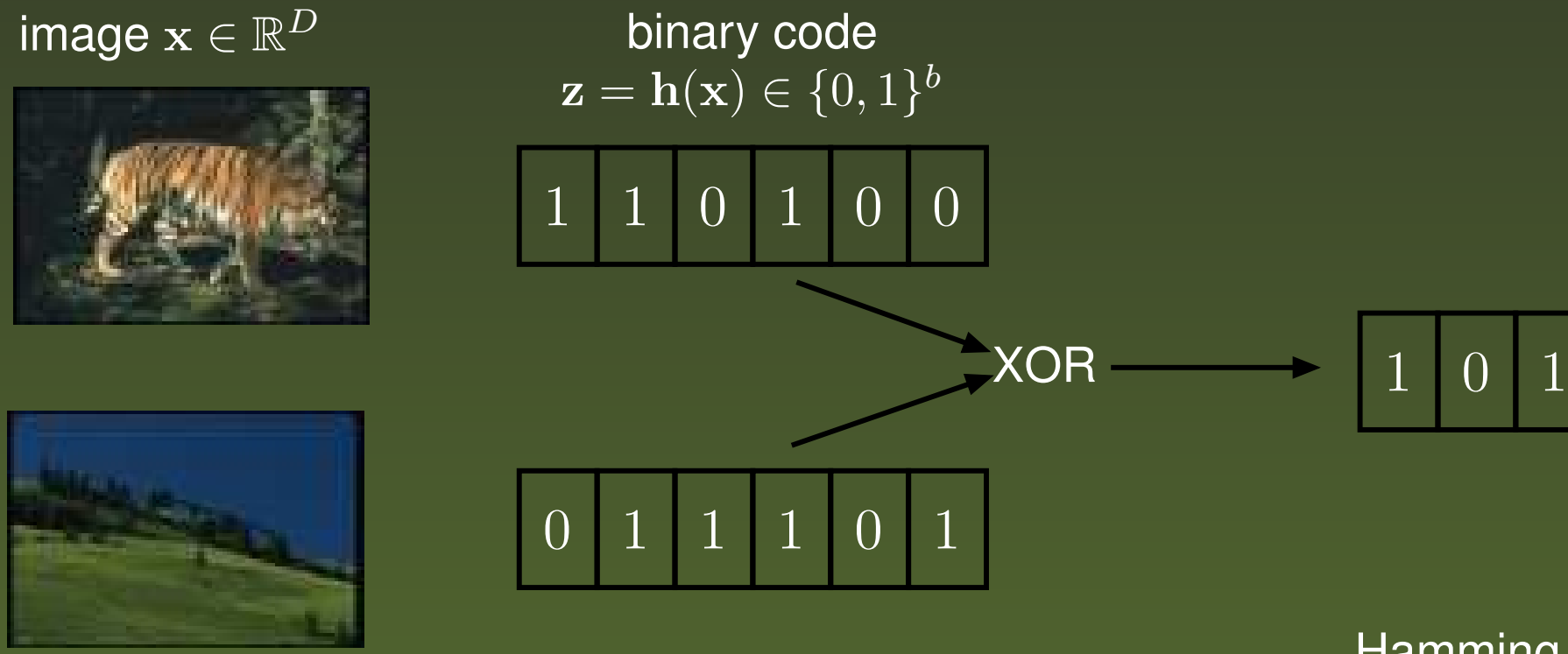


A fast, approximate approach: binary hashing.

Large scale image retrieval: binary hash functions

A binary hash function $\mathbf{h}$ maps a high-dimensional vector $\mathbf{x} \in \mathbb{R}^D$ to a b-bit vector $\mathbf{z} = \mathbf{h}(\mathbf{x}) = (h_1(\mathbf{x}), \ldots, h_b(\mathbf{x})) \in \{0,1\}^b$. It should:

- preserve neighbours: map (dis)similar images to (dis)similar codes (in Hamming distance)
- be fast to compute.



Large scale image retrieval: binary hash functions

Scalability: dataset with millions or billions of high-dimensional images.

- Time complexity: $\mathcal{O}(Nb)$ instead of $\mathcal{O}(ND)$ with small constants.

  Bit operations to compute Hamming distances instead of floating point operations to compute Euclidean distances.

- Space complexity: $\mathcal{O}(Nb)$ instead of $\mathcal{O}(ND)$ with small constants.

  We can fit the binary codes of the entire dataset in faster memory, further speeding up the search.

Ex: $N = 10^6$ points, $D = 300$ and $b = 32$:

	space	time
Original space	1.2 GB	20 ms
Hamming space	4 MB	30 μs

We need to learn the binary hash function $\mathbf{h}$ from a training set.
Ideally, we'd optimise precision/recall directly, but this is difficult.
Instead, one often optimises a proxy objective, usually derived from dimensionality reduction.

Supervised hashing: similarity-based objective function

A similarity matrix $\mathbf{W}$ determines similar and dissimilar pairs of points among the points in the training set $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$, for example:

$$w_{nm} = \begin{cases} +1 & \mathbf{x}_n \text{ and } \mathbf{x}_m \text{ are similar} \\ -1 & \mathbf{x}_n \text{ and } \mathbf{x}_m \text{ are dissimilar} \\ 0 & \text{we do not know.} \end{cases}$$

Then we learn the b-bit hash function $\mathbf{h}$: $\mathbb{R}^D \rightarrow \{-1, +1\}^b$ by minimising an objective function based on $\mathbf{W}$, e.g. the Laplacian loss:

$$\mathcal{L}(\mathbf{h}) = \sum_{n,m=1}^{N} w_{nm} \left\| \mathbf{h}(\mathbf{x}_n) - \mathbf{h}(\mathbf{x}_m) \right\|^2 \quad \text{s.t.} \quad \mathbf{h}(\mathbf{X})^T \mathbf{h}(\mathbf{X}) = N\mathbf{I}_b.$$

The objective tries to preserve the point neighbourhoods and the constraints make the single-bit functions differ from each other.

While we focus on Laplacian loss for simplicity, other loss functions can also be used (KSH, BRE, etc.).

The hash function is typically a thresholded linear function.

Optimisation-based approaches

Much binary hashing work has studied how to optimise this problem.

- Relaxation (e.g. Liu et al., 2012): relax the step function or binary codes (ignoring the binary nature of the problem), optimise the objective continuously and truncate the result.
- Two-step methods (Lin et.al., 2013, 2014): first, define the objective over the binary codes and optimise it approximately; then, fit the hash function to these the codes.
- Method of auxiliary coordinates (R. & C.-P., NIPS 2016): this achieves the lowest objective value by respecting the binary nature of the problem and optimising the codes and the hash function jointly.

Limitations: difficult, slow optimisation:

- Nonconvex, nonsmooth: the hash function outputs binary values.
  Underlying problem of finding the binary codes is an NP-complete optimisation over Nb variables.
- The b single-bit hash functions are coupled.
  To avoid trivial solutions where all codes are the same.
- Slow optimisation, doesn't scale beyond a few thousand points.
- Optimising the objective very accurately helps, but doesn't seem to produce a much better precision/recall.

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

An ensemble diversity approach (Carreira-Perpiñán & Raziperchikolaei, NIPS 2016)

It is possible to learn a very good hash function $\mathbf{h}\colon \mathbb{R}^D \to \{-1,+1\}^b$ by simply optimising each of the b single-bit hash functions $h_1(\mathbf{x}),\dots,h_b(\mathbf{x})$ independently of the others, and making them diverse by other means, not optimisation-based.

Independent Laplacian Hashing (ILH): optimise the single-bit objective b times independently to obtain $h_1(\mathbf{x}),\dots,h_b(\mathbf{x})$:

$$\mathcal{L}(h) = \sum_{n,m=1}^{N} w_{nm}(h(\mathbf{x}_n)-h(\mathbf{x}_m))^2 \qquad h\colon \mathbb{R}^D \to \{-1,+1\}.$$

An additional consequence: while in the b-bit case there exist many different objective functions, they all become essentially identical in the $b=1$ case, and have the form of a binary quadratic function (a Markov random field) $\min_{\mathbf{z}} \mathbf{z}^T\mathbf{A}\mathbf{z}$ with $\mathbf{z}\in\{-1,+1\}^N$ for a certain matrix $\mathbf{A}_{N\times N}$:

Objective $\mathcal{L}(\mathbf{h})$	b-bit	1-bit
KSH	$(\mathbf{z}_n^T\mathbf{z}_m - bw_{nm})^2$	$-2w_{nm}z_nz_m+$ constant
BRE	$\left(\frac{1}{b}\left\|\mathbf{z}_n-\mathbf{z}_m\right\|^2 - w_{nm}\right)^2$	$-4(2-w_{nm})z_nz_m+$ constant
Laplacian	$w_{nm}\left\|\mathbf{z}_n-\mathbf{z}_m\right\|^2$	$-2w_{nm}z_nz_m+$ constant

An ensemble diversity approach (Carreira-Perpiñán & Raziperchikolaei, NIPS 2016)

If we optimise the same objective function b times, we get b identical hash functions and we gain nothing over a single hash function.

How to make sure that the b hash functions are different from each other and their combination results in good retrieval?

ILH uses diversity techniques from the ensemble learning literature:

- Different training sets (ILHt):
  Each hash function uses a training set different from the rest.
  Sampled randomly from the available training data.
- Different initializations (ILHi):
  Each hash function is initialised randomly.
- Different feature subsets (ILHf):
  Each hash function is trained on a random subset of features.

Of these, ILHt works best in practice, and we focus on it.

Advantages of Independent Laplacian Hashing (ILH)

Learning the b single-bit hash functions independently is simple and works well:

- Most importantly, and perhaps surprisingly, ILH is better than or comparable to the optimisation-based methods in retrieval tasks, particularly as one increases the number of bits b.
- Much simpler and faster optimisation.

  b independent problems each over N binary codes rather than 1 problem with Nb binary codes.
- Training the b hash functions is embarrassingly parallel.
- ILH can scale to larger training sets per bit, and overall use more training data that optimisation-based approaches.

  We can easily use millions of points in learning the hash functions.
- To get the solution for $b+1$ bits we just need to take a solution with b bits and add one more bit, which is helpful for model selection.

In this paper, we propose two simple but effective improvements to ILH.

1. Pruning a set of hash functions: ILH-prune

Given a set of b single-bit hash functions, we want to select a subset of $s < b$ hash functions which performs comparably well in a retrieval task, but is therefore faster at run time.

This is possible because some hash functions may be redundant or ineffective.

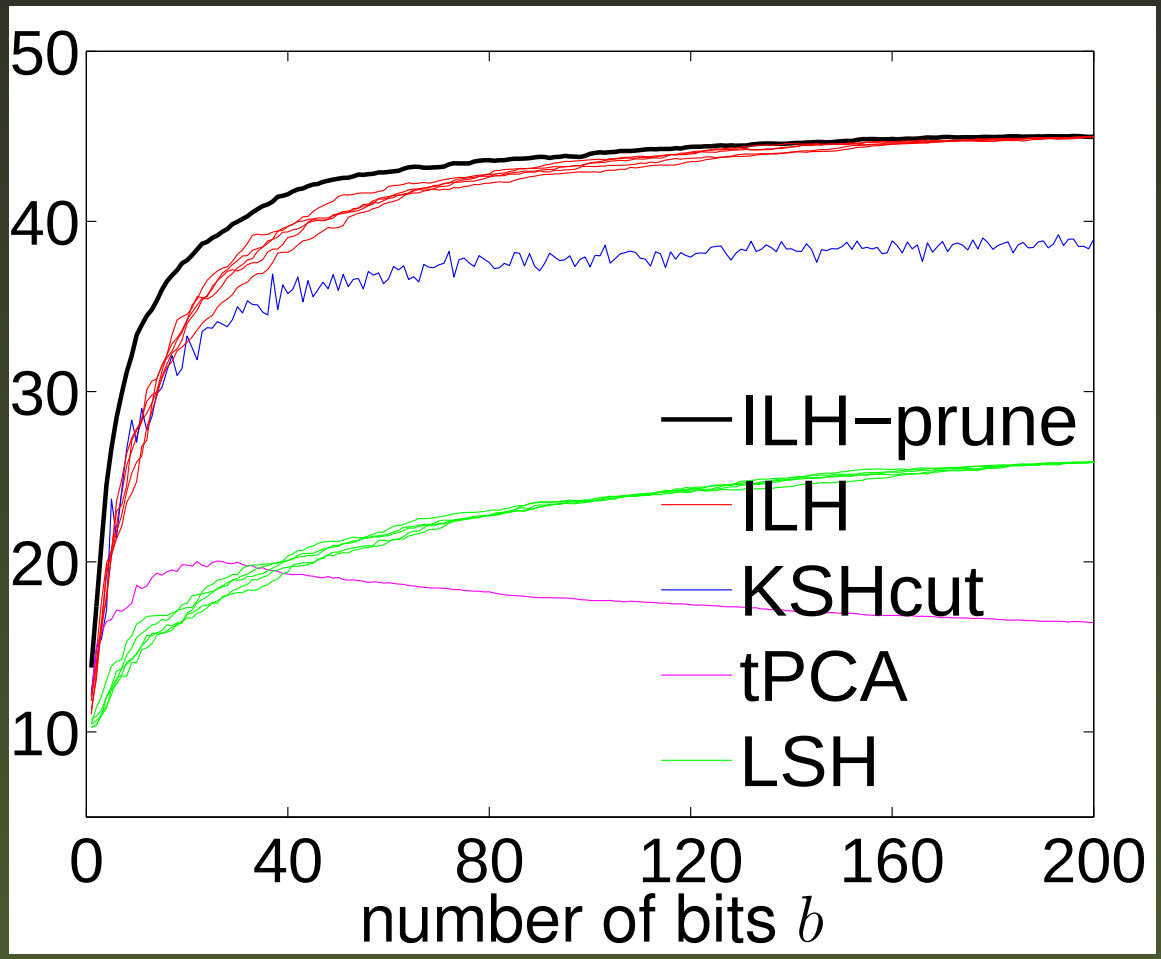
We seek the subset of hash functions that maximises the precision on a given test set of queries. A brute-force search is impractical because there are $\binom{b}{s}$ subsets. We solve this combinatorial problem approximately with a greedy algorithm, sequential forward selection:

- Starting with an empty set, repeatedly add the hash function that, when combined with the current set, gives highest precision.
- Stop when we reach a user-set value for:
  - the number s of functions, or...
  - the percentage of the precision of the entire set of b functions.

Pruning can be applied to post-process the hash functions of any method, not just ILH, such as optimisation-based approaches.

ILH-prune: precision as a function of the number of bits

CIFAR dataset, $N = 58\,000$ training / $2\,000$ test images, $D = 320$ SIFT features.

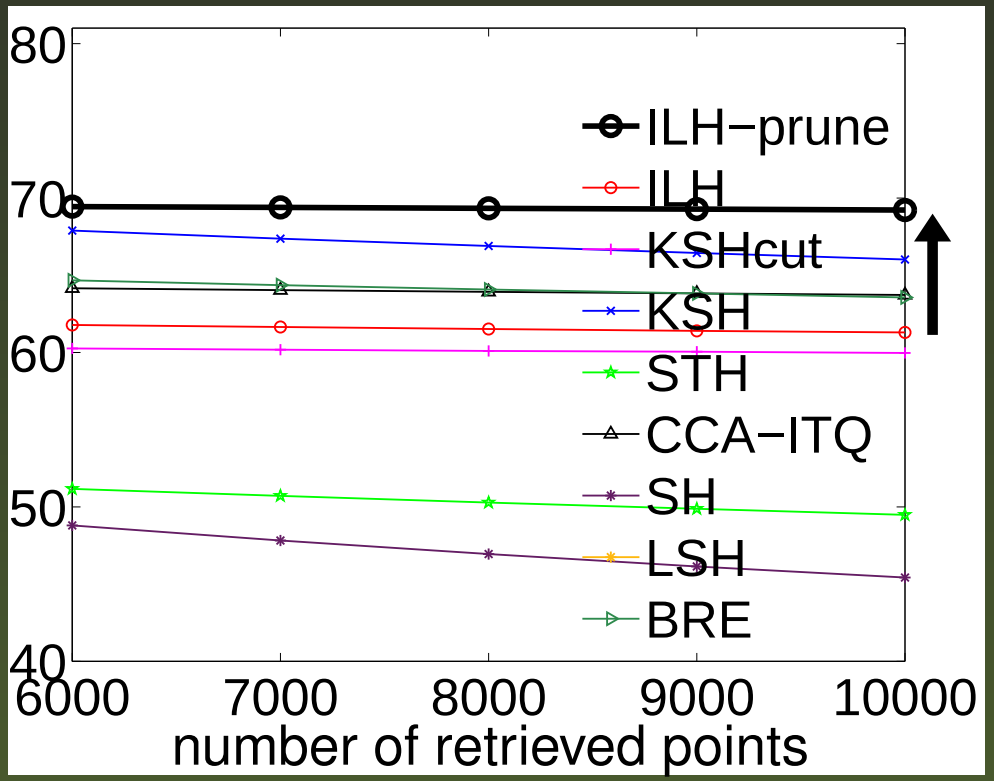


ILH-prune achieves nearly the same precision as ILH but with a quite smaller number of bits.

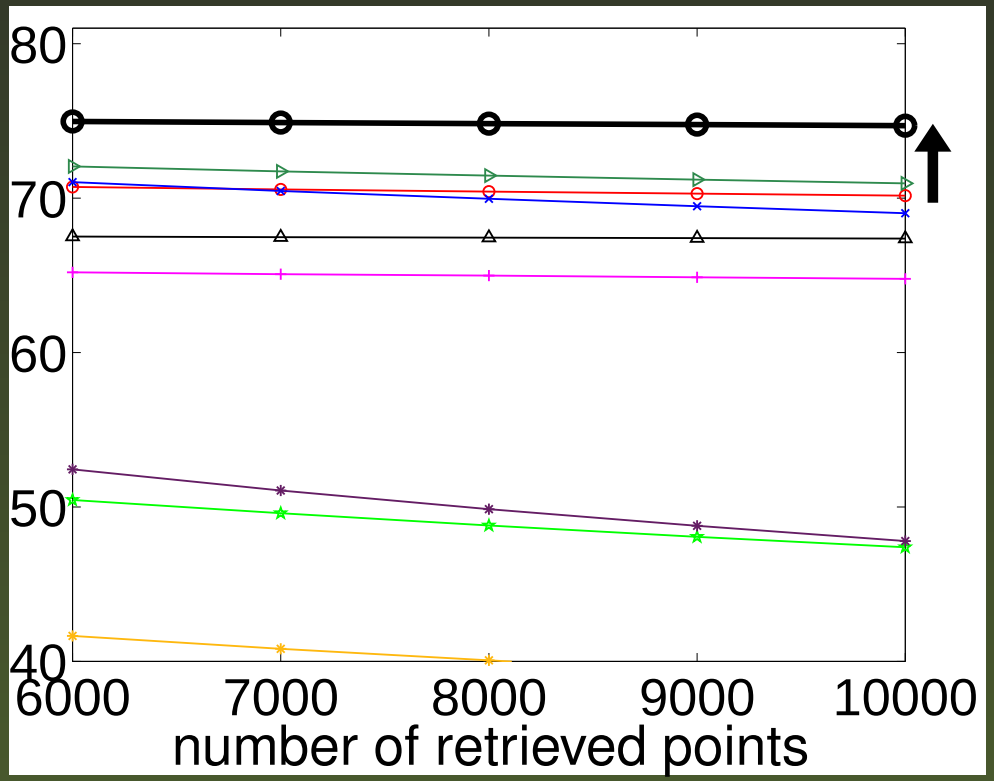
ILH-prune compared with other hashing methods

Infinite MNIST dataset, $N = 1\,000\,000$ training / $2\,000$ test images, $D = 784$ vector of raw pixels.
Ground-truth: points with the same label as the query.

$b = 16$

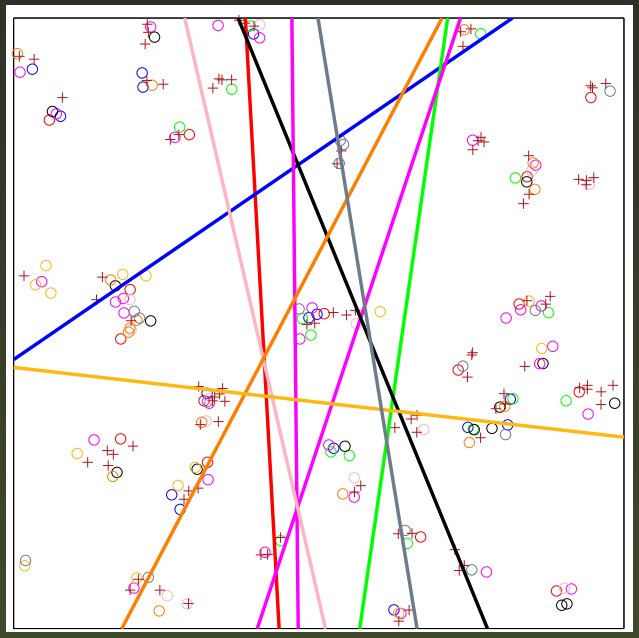


$b = 32$

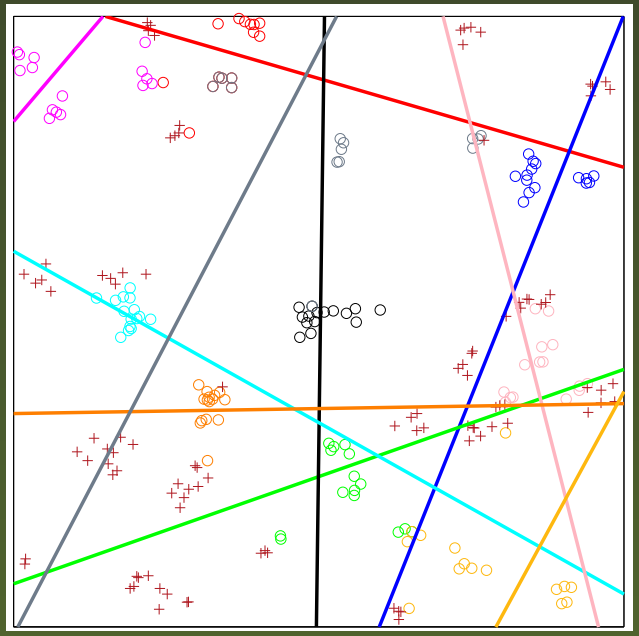


ILH beats all methods as the number of bits b increases, but not always if using a small b. With pruning, it is also the best method with small b.

2. Learning the hash functions locally: ILH-local



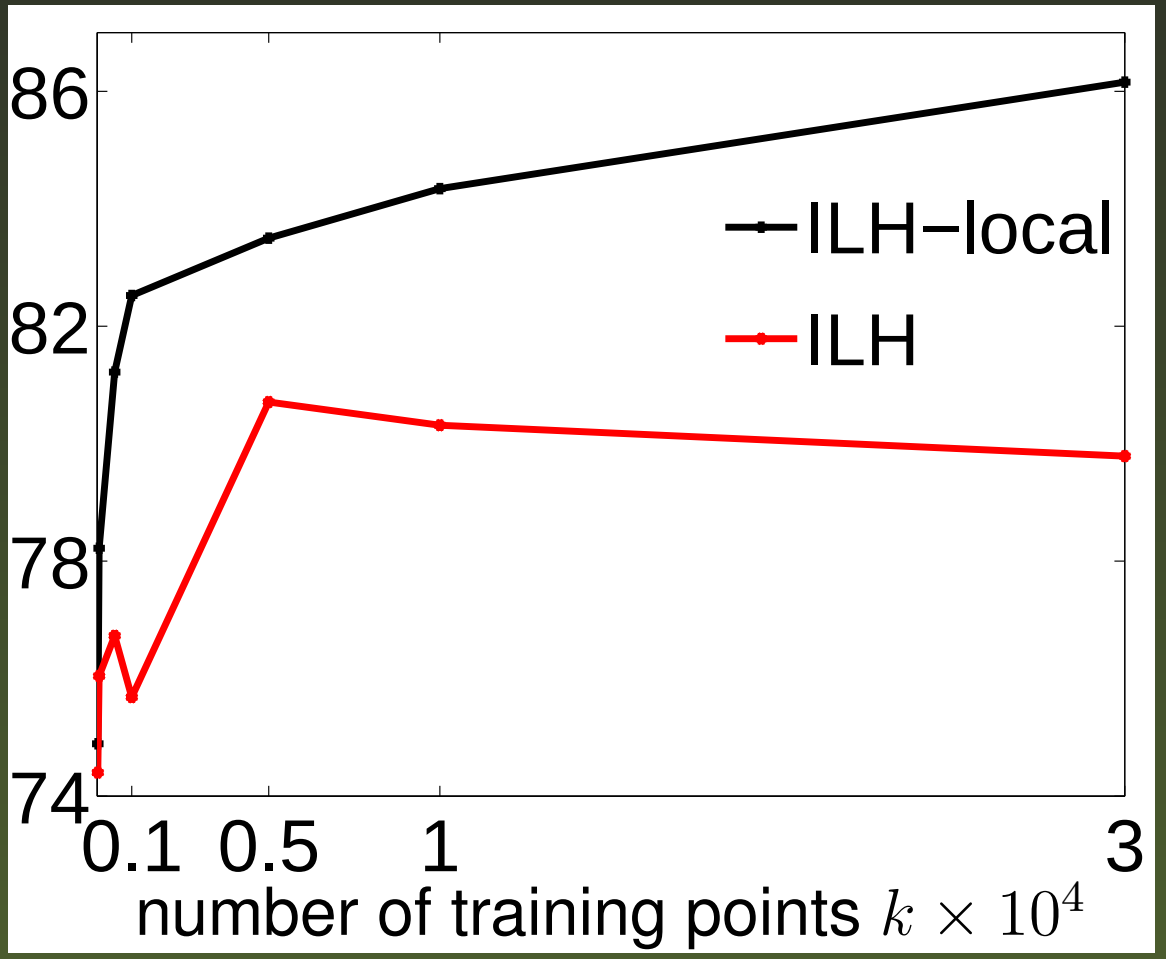
ILH: the training subsets for the b single-bit hash functions span the entire input space and have high overlap spatially. This can decrease the resulting diversity and make some of the single-bit hash functions be very similar to each other, hence resulting in a lower precision.



ILH-local avoids this by selecting spatially local subsets. It defines the training subset for a given single-bit hash function as a training point $\mathbf{x}_n$ (picked at random) and its k nearest neighbors. This improves the diversity and neighbourhood preservation, hence resulting in a higher precision.

ILH-local: precision by changing size of the training set

Infinite MNIST dataset, $N = 1\,000\,000$ training / $2\,000$ test images, $D = 784$ vector of raw pixels.
Ground-truth: points with the same label as the query. We use $b = 200$ bits.

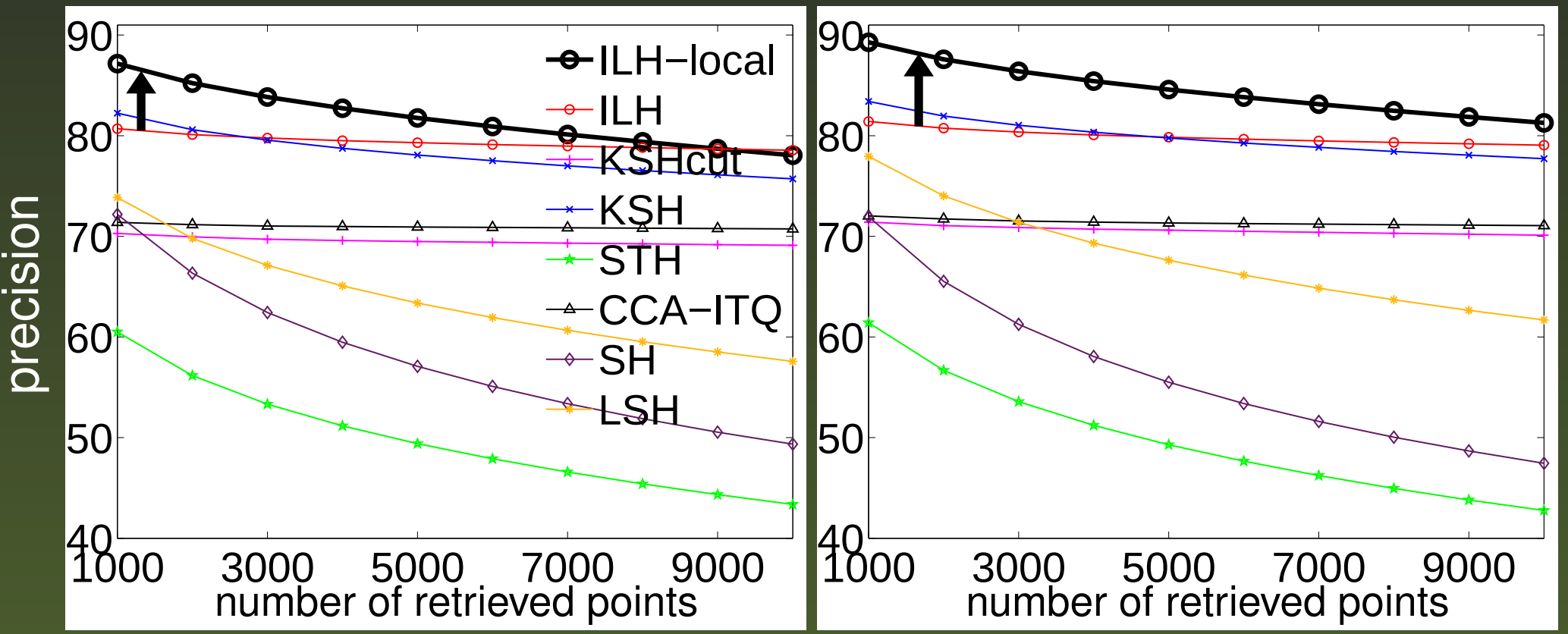


ILH-local performs better than ILH over the entire range of training set sizes.

ILH-local compared with other hashing methods

Infinite MNIST dataset, $N = 1\,000\,000$ training / $2\,000$ test images, $D = 784$ vector of raw pixels.
Ground-truth: points with the same label as the query.

$b = 128$ $b = 200$



ILH-local improves the results of ILH significantly and beats state-of-the-art methods.

Conclusion

- Most hashing papers use an optimisation-based approach to learn hash functions, which couples all the single-bit functions. This results in a very difficult, slow optimisation.
- A different approach that works as well or better in terms of retrieval performance is to train the single-bit hash functions independently but make them diverse by training them on different data subsets, as done by independent Laplacian hashing (ILH).
- We improve the results of ILH by pruning and locality techniques:
  - By using forward selection, we can prune a large set of single-bit hash functions and achieve comparable results using a small number of bits.
  - By selecting the training points for each hash function of ILH locally in input space, we learn more diverse hash functions that achieve higher precision.

Code available at the authors' web page. Partly supported by NSF award IIS–1423515.