An Ensemble Diversity Approach to Binary Hashing

Ramin Raziperchikolaei

Electrical Engineering and Computer Science

University of California, Merced

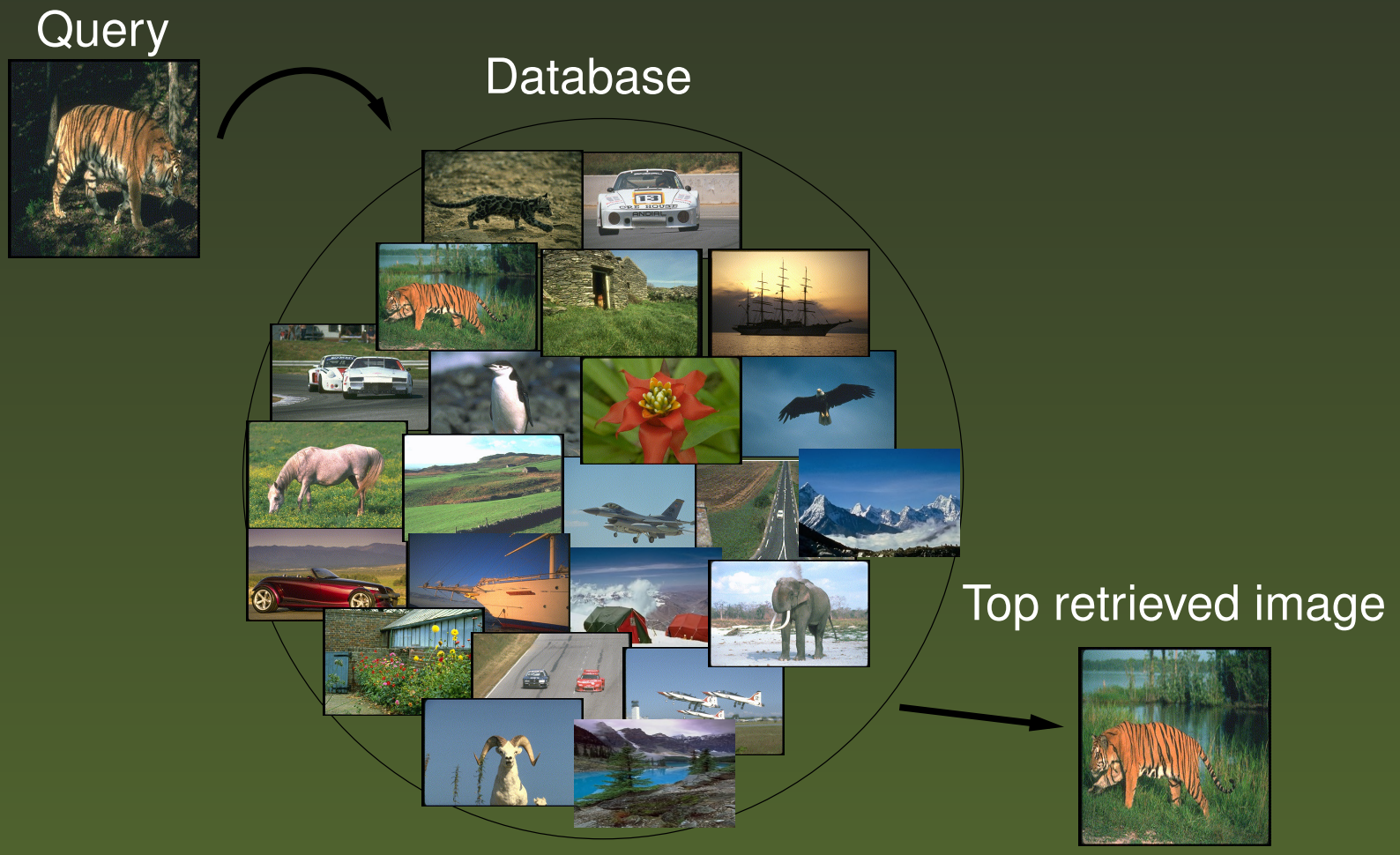
http://eecs.ucmerced.edu

Joint work with **Miguel Á. Carreira-Perpiñán**

Large Scale Image Retrieval

Searching a large database for images that are closest to a query.

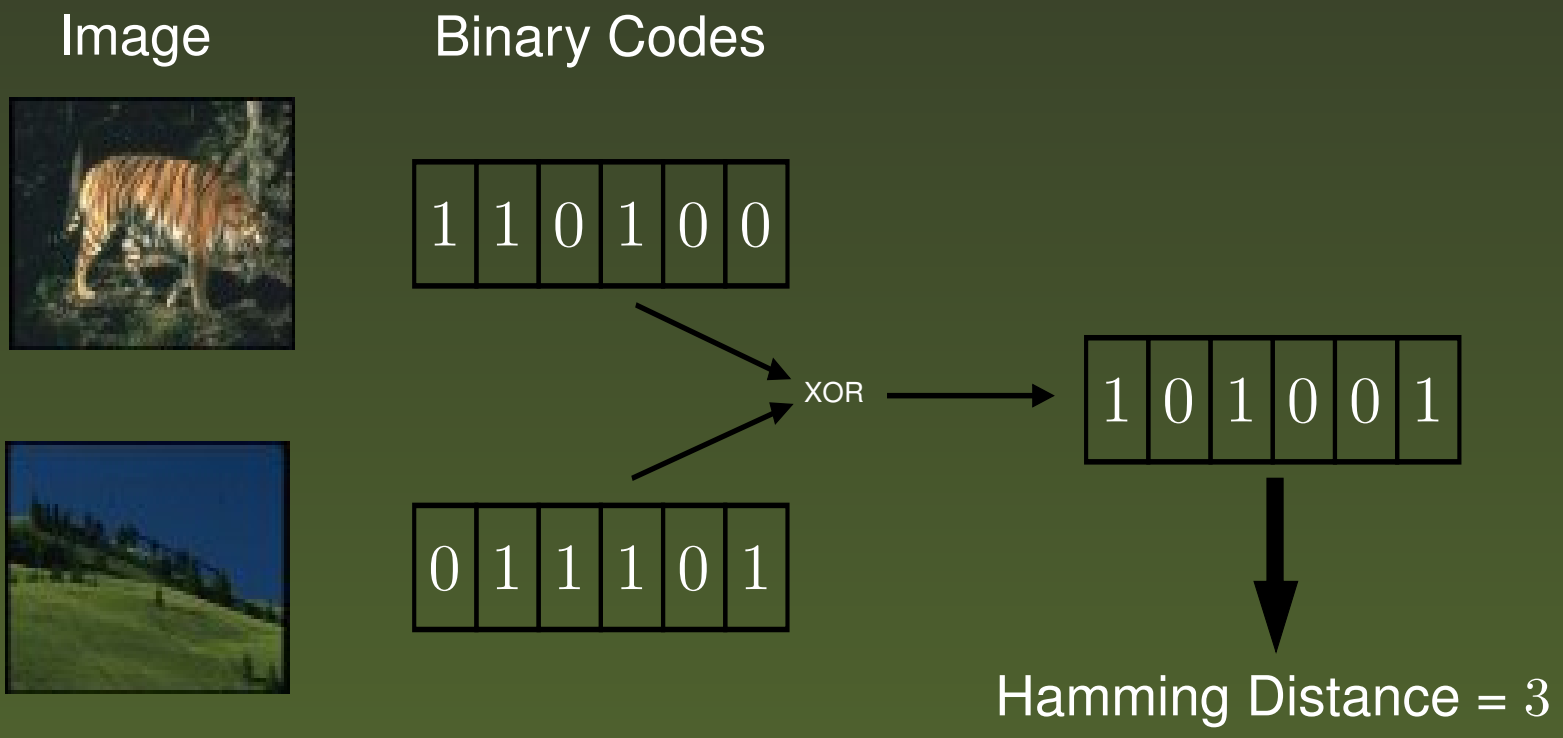
This is the k nearest neighbors problem on N vectors in $\mathbb{R}^D$ with large N and D.



Binary Hash Functions

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 $\mathbf{z} = \mathbf{h}(\mathbf{x}) \in \{0, 1\}^b$.

- Main goal: preserve neighbors, i.e., assign (dis)similar codes to (dis)similar patterns.
- Hamming distance computed using XOR and then counting.



Binary Hash Functions in Large Scale Image Retrieval

Scalability: we have 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 distance instead of floating point operations to compute Euclidean distance.
- Space complexity: $\mathcal{O}(Nb)$ instead of $\mathcal{O}(ND)$ with small constants.

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

Ex: $N = 1\,000\,000$ points, $D = 300$ and $b = 32$:

	Space	Time
Original space	1.2 GB	20 ms
Hamming space	4 MB	30 μs

Affinity-Based Objective Functions

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

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

$\mathbf{x}_n$ and $\mathbf{x}_m$ are similar if: $\begin{cases} \text{label}(\mathbf{x}_n) = \text{label}(\mathbf{x}_m) & \text{supervised dataset} \\ \|\mathbf{x}_n - \mathbf{x}_m\| < \epsilon & \text{unsupervised dataset} \end{cases}$

Learn hash function $\mathbf{h}(\cdot) \in \{0,1\}^b$ by minimizing the affinity-based objective function:

$$\min \mathcal{L}(\mathbf{h}) = \sum_{n,m=1}^{N} L(\mathbf{h}(\mathbf{x}_n), \mathbf{h}(\mathbf{x}_m);\ w_{nm}) \quad \text{where} \quad \mathbf{h}(\mathbf{x}_n) \in \{0,1\}^b$$

$L(\cdot)$ is a loss function that compares the codes for two images with the ground-truth value w_{nm}.

Optimizing Affinity-Based Objective Functions

Many hashing papers use affinity based objective function:

Laplacian loss (Spectral Hashing (Weiss et al. 2008), Hashing with Graphs (Liu et al. 2011), etc.):

$$\mathcal{L}(\mathbf{h}) = \sum_{n,m=1}^{N} w_{nm} \left\|\mathbf{h}(\mathbf{x}_n) - \mathbf{h}(\mathbf{x}_m)\right\|^2 \text{ s.t. } \begin{array}{l} \mathbf{h}(\mathbf{X})^T\mathbf{h}(\mathbf{X}) = N\mathbf{I} \\ \mathbf{h}(\mathbf{X})^T\mathbf{1} = 0. \end{array}$$

KSH Loss (Supervised Hashing with Kernels (Liu et al. 2012), Two-Step Hashing (Lin et al. 2013), etc.):

$$\mathcal{L}(\mathbf{h}) = \sum_{n,m=1}^{N} \left(\mathbf{h}(\mathbf{x}_n)^T\mathbf{h}(\mathbf{x}_m) - b\, w_{nm}\right)^2$$

Since the output of the hash function is binary, the objective function is nonsmooth and difficult to optimize.

All the one bit hash functions, $\mathbf{h} = [h_1, \ldots, h_b]$, are coupled to force them to be different from each other. This further complicates the optimization: Optimization takes a long time, it limits the number of points and bits in training, etc.

The goal of Most binary hashing wroks is to propose a new objective function and an approximate way to optimize it. We propose a different approach to learn good hash functions.

Training Binary Hash Functions Independently

We propose to optimize each 1-bit hash function independently from the rest.

This gives us several advantages:

- Optimization simplifies greatly: we deal with b independent problem each over N binary codes rather than 1 problem with with Nb binary codes.
- This will lead to faster training and better accuracy.
- Training can be done in parallel.

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

We will introduce diversity in a different way:
We use diversity techniques from the ensemble learning literature.

A Single Bit Affinity-based Objective Function

Independent Laplacian Hashing (ILH):

We focus on the following objective function to learn a 1-bit hash function $h(\cdot)$:

$$\mathcal{L}(h) = \sum_{n,m=1}^{N} w_{nm}(h(\mathbf{x}_n) - h(\mathbf{x}_m))^2$$

We can use existing algorithms for optimizing affinity-based objective functions, which becomes particularly effective with our 1-bit objective functions. For example:

- (1) Relax the binary constraints, (2) solve the problem assuming that the hash functions are continuous and (3) truncate the results to achieve the binary codes.
- (1) Replace the hash functions by binary codes $z_n = h(\mathbf{x}_n)$, (2) find the binary codes using binary optimization techniques like Graph-Cut, and (3) Learn hash functions by training classifiers from input to the binary codes.

We show that we can avoid trivial solutions by injecting diversity into each hash function's training using techniques inspired from classifier ensemble learning.

Adding Diversity with Ensemble Learning Techniques

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

A similar problem arises in ensemble learning: we want to train several classifiers on the same training set. If the classifiers are all equal, we gain nothing over a single classifier.

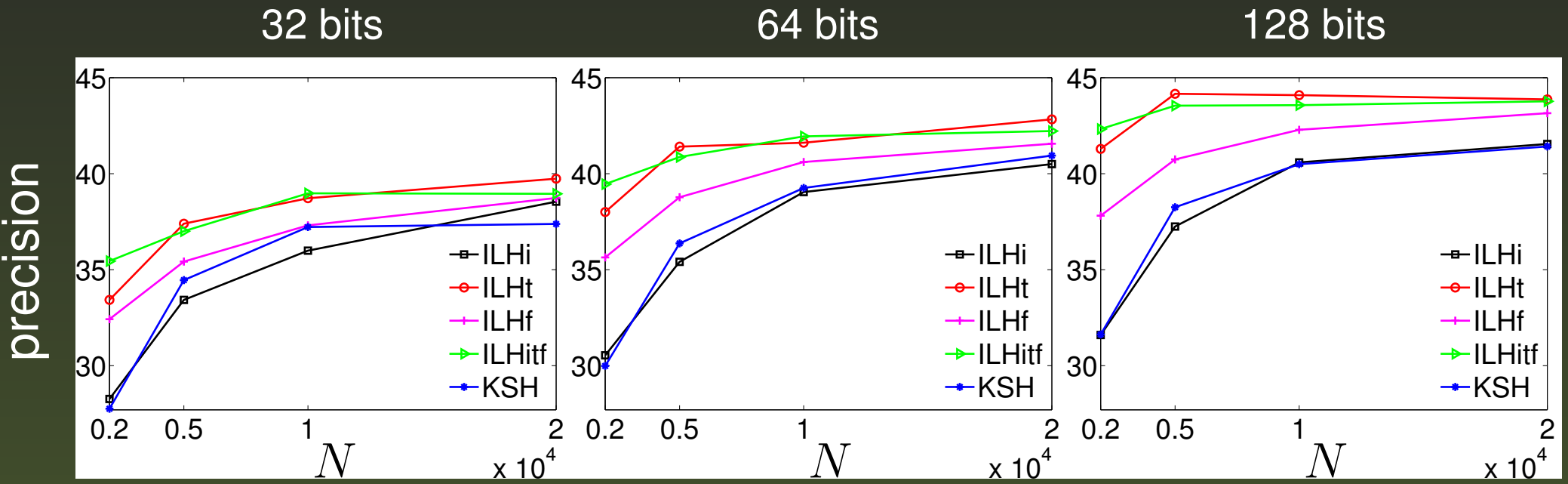
We consider the following diversity mechanisms from the ensemble learning literature:

- 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 $1 \leq d \leq D$ features.

Advantages of Independent Laplacian Hashing

- b binary optimizations over N binary variables each is generally easier than one binary optimization over bN variables.
- Training the b functions can be parallelized perfectly.
- To get the solution for $b+1$ bits we just need to take a solution with b bits and add one more bit.
  - This is helpful for model selection. How many bits do we need in binary hashing? We can maximize the precision on a test set over b (cross-validation).
  - Computationally easy: simply keep adding bits until the test precision stabilizes.
- For ILHf, both the training and test time are lower than if using all D features for each hash function. The test runtime for a query is d/D smaller.

Experiments: Diversity Mechanisms with ILH

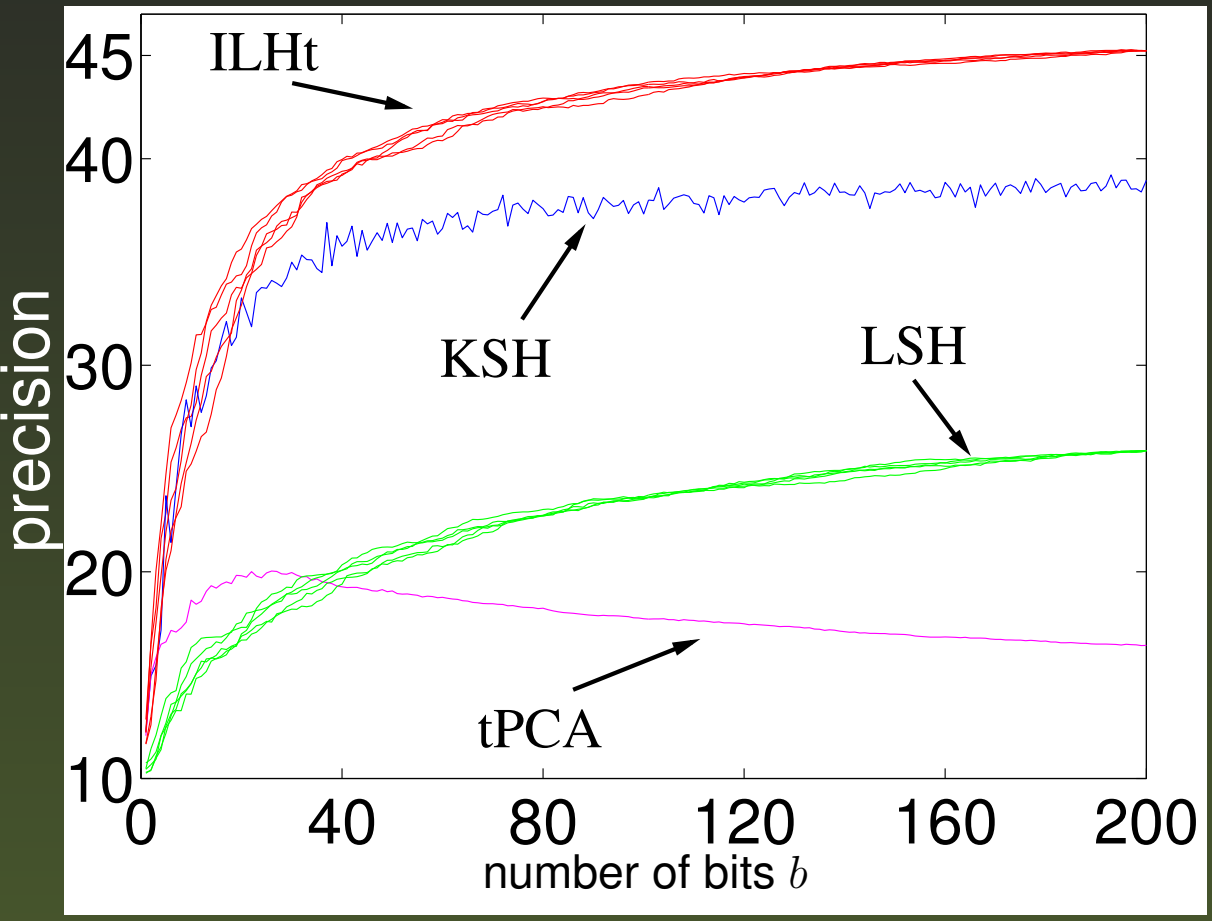


CIFAR dataset, $N = 58\,000$ training/ $2\,000$ test images, $D = 320$ SIFT features.

As a device to make the hash functions different and produce good retrival, the diversity mechanisms work as well as or quite better than using optimization.

The clearly best diversity mechanism is ILHt, which works better than the other mechanisms, even when combined with them, and significantly better than KSH.

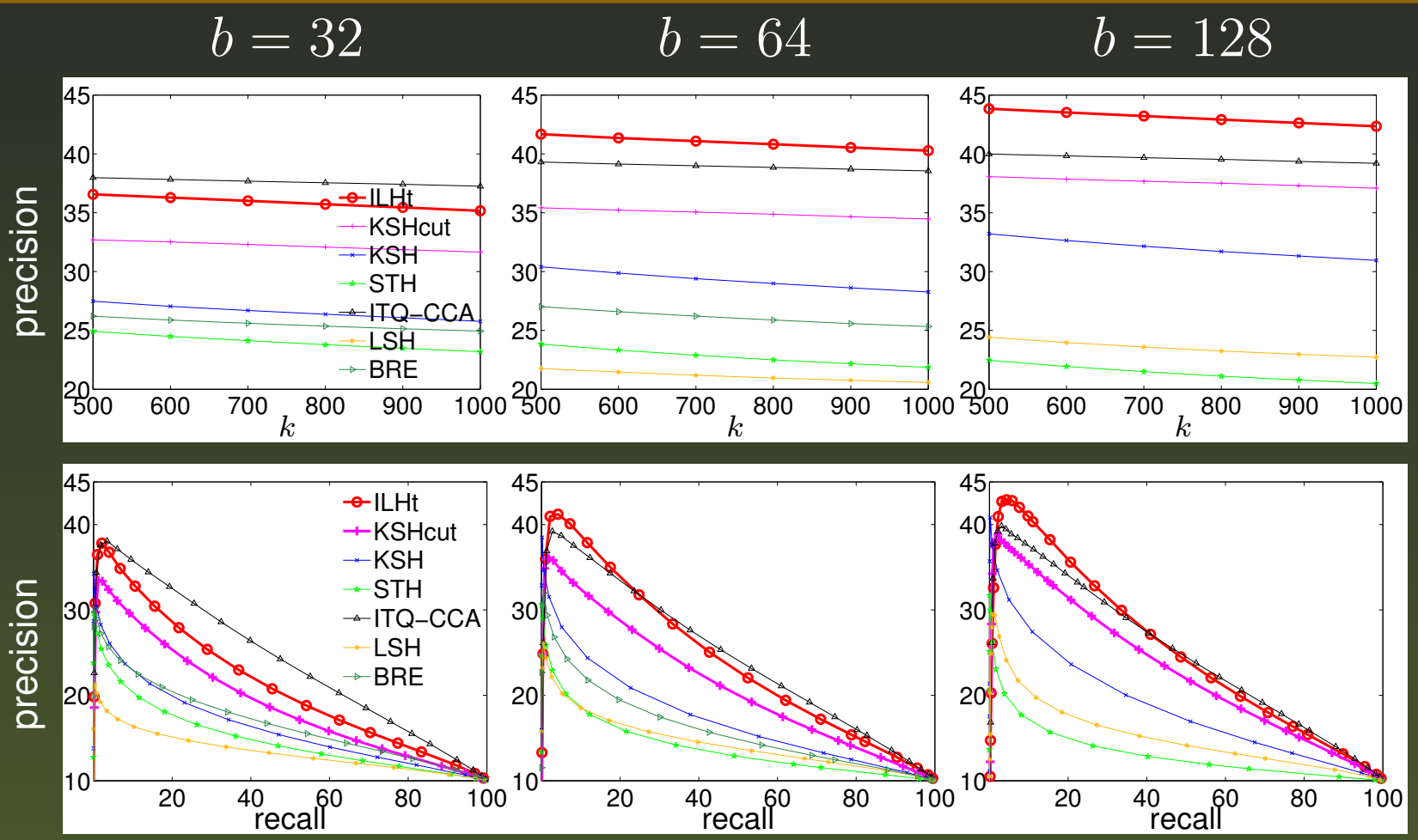
Preformance as a Function of Number of Bits



For KSH the variance is large (compared to ILHt) and the precision barely increases after $b = 80$.

For ILHt, the precision increases nearly monotonically and continues increasing beyond $b = 200$ bits.

ILHt Compared with Other Binary Hashing Methods



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

Groundtruth: points with the same labels as the query

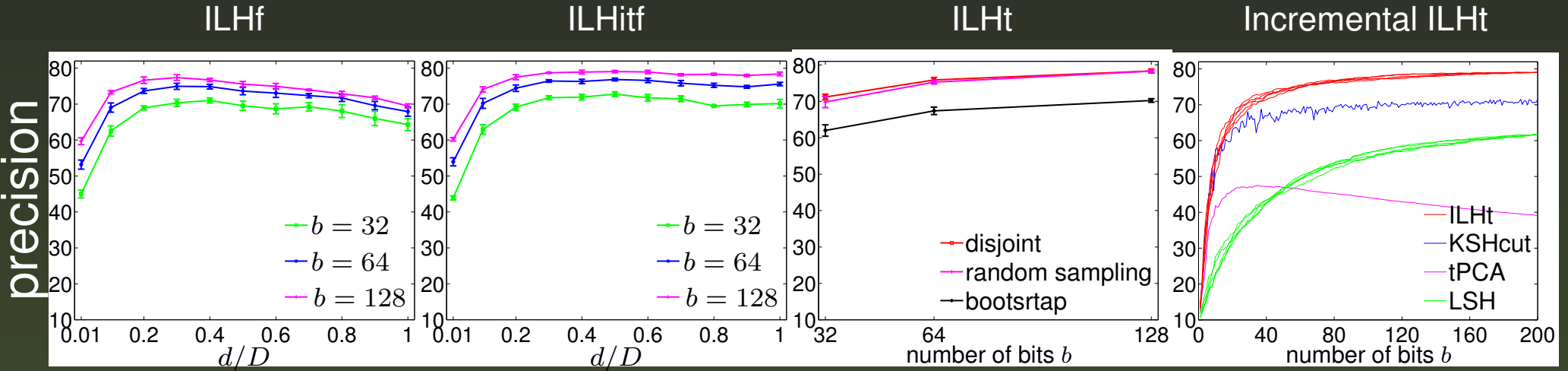
ILHt beats state-of-the-art methods, particularly as the number of bits b increases.

Conclusion

- Most hashing papers try to learn good hash functions by minimizing a sophisticated affinity-based objective function that couples all the binary codes. This results in a very difficult, slow optimization.
- This is not necessary! We have shown that the hash functions can be trained independently:
  - Much simpler optimization. Over N binary codes instead of Nb.
  - Training is fast and parallel. b 1-bit hash functions trained independently.
  - Performance is competitive or even quite better than the state-of-the-art.
- We need diversity techniques to avoid trivial solutions:
  - ILHi: different initialization.
  - ILHf: different sets of features in training hash functions.
  - ILHt: different subsets of points and works best.

Partly supported by NSF award IIS–1423515.

Experiments: Diversity Mechanisms with ILH (Cont.)



INFMNIST dataset, $N = 1\,000\,000$ training/ $2\,000$ test images, $D = 784$ raw pixel features.

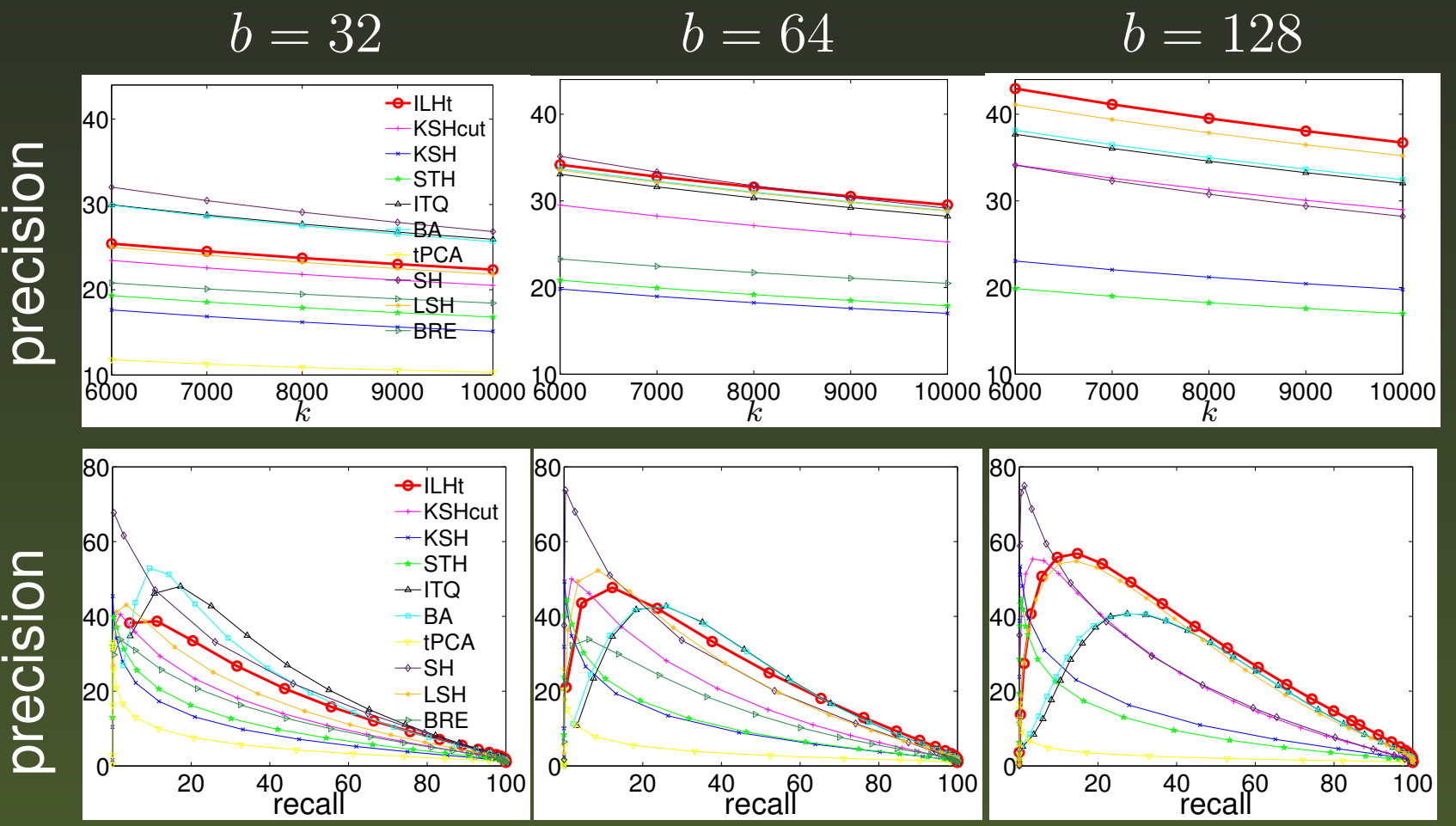
(panels 1–2) shows the results in ILHf of varying the number of features $1 \leq d \leq D$ used by each hash function. The highest precision is achieved with a proportion $d/D \approx 30\%$ for ILHf.

(panel 3) shows the results of using bootstrapped (samples with replacement from $5\,000$ points) instead of disjoint training sets for ILHt. As expected, the latter is consistently better.

(panel 4) shows the precision (in the test set) as a function of the number of bits b for ILHt, where the solution for $b + 1$ bits is obtained by adding a new bit to the solution for b.

- For KSHcut the variance is large (compared to ILHt) and the precision barely increases after $b = 30$.
- For ILHt, the precision increases nearly monotonically and continues increasing beyond $b = 200$ bits.

Comparison with Other Binary Hashing Methods



FLICKR dataset, $N = 1\,000\,000$ training/ $10\,000$ test images, $D = 150$ edge histogram features.

Groundtruth: First $K = 10\,000$ nearest neighbors of the query in the original space.

ILHt beats state-of-the-art methods, particularly as the number of bits b increases.