ΠΕΤΡΑΚΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ AM: 1041589 ΥΔΑ

Scalable Supervised Discrete Hashing for Large Scale Search

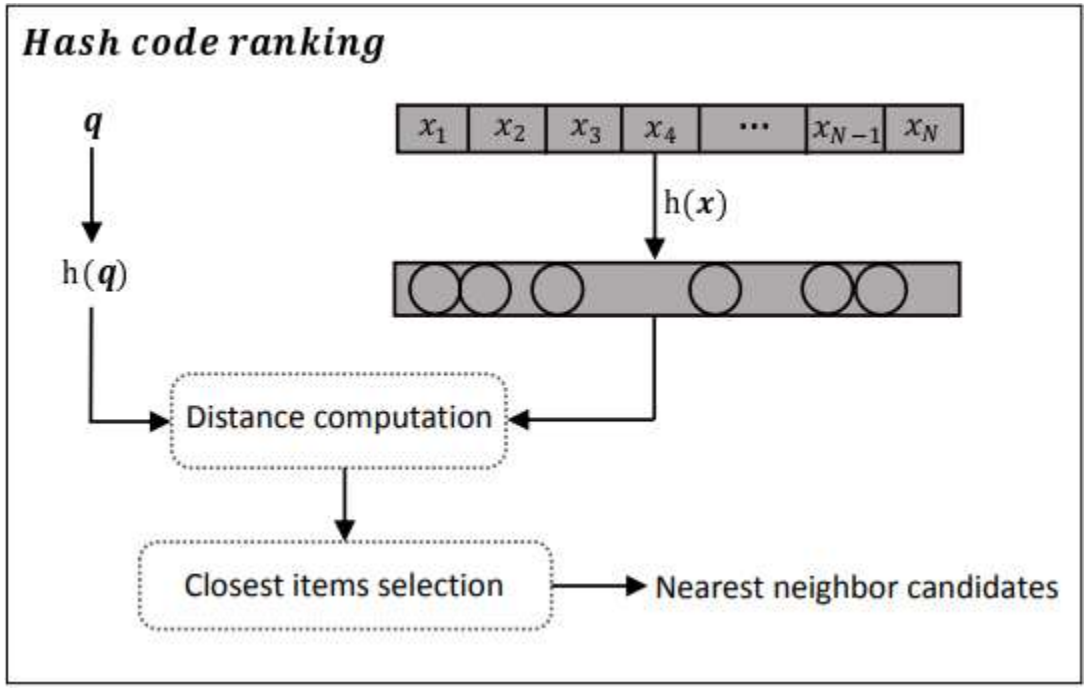
XIN LUO, YE WU, XIN-SHUN XU

Problem at hand

- Many important applications utilize the similarity search technique, which can return similar data instances to a given query instance
- As the amount of data increases explosively the exhaustive comparisons among the query instance and candidate ones in original high dimensional space make traditional similarity search methods inappropriate for scalable search
- The solution for fast similarity search: Hashing methods
  - Recommendation, retrieval, and multimedia analysis
    - web image retrieval, mobile landmark search, video retrieval, recommendation, person re-identification, sequential or online data search, cross-modal or cross-view retrieval, search in a distributed setting

Hashing

- Hashing methods learn the hash function to transform original data into compact binary hash codes
- Storage cost can be highly reduced
  - By storing the hash codes instead of the original high-dimensional data
- When queries come, they will first be transformed into binary hash codes by the learnt function. Then, similarity search can be simply done by calculating the Hamming distances between the hash codes of available data instances and queries
  - the Hamming distance between two binary codes is simply the number of bits that differ
    - It can be calculated using the bitwise operation XOR
    - the retrieval process can be efficiently conducted



Data-independent vs Data-dependent Hashing

- Data-independent methods
  - do not consider the specific data, and leverage random projections to learn the hash function (e.g. LSH)
  - main drawback: they need long codes to guarantee high precision,
    - huge storage overhead
- Data-dependent
  - take the specific data into consideration and learn compact binary codes, which can effectively and highly efficiently index and organize massive data
  - Unsupervised
  - Supervised

Unsupervised vs Supervised Hashing

- Unsupervised
  - do not leverage the label/semantic information and learn hash codes or hash functions through exploiting the relations of training data
  - e.g. Spectral Hashing, Iterative Quantization, Inductive Hashing on Manifolds
- Supervised
  - When the label information is available, supervised hashing methods can embed the semantic information into the learning of hash codes
  - They have demonstrated better performance in many real-world applications
  - e.g. Semi-Supervised Hashing, Minimal Loss Hashing, Two-Step Hashing, Supervised Hashing with Latent Factor

Problems of Supervised Hashing methods

- The binary constraints of hash codes lead to a discrete optimization problem which is hard to solve
  - most methods relax the discrete constraints for easy optimization
  - then quantize the learnt real-valued solution to binary hash codes
  - the solution may be sub-optimal and the error caused by relaxation usually degrades the performance
- As the supervised methods need to embed the label information, most of them construct a $n \times n$ instance-pairwise similarity matrix which is leveraged in the learning procedure
  - Space Cost : O(n^2)
- Thus, these hashing methods are time-consuming and unscalable
  - To tackle this, usually they sample a small subset of instances for training and discard the others
    - This may cause information loss and result in poor performance.

Scalable Supervised Discrete Hashing

- It can discretely and efficiently learn the hash codes by making full use of all training samples and semantic information
- It takes both the pairwise similarity **S** and the label matrix **Y** into consideration
  - this ensures more precise hash codes
- They represent the hash codes matrix **B** with a real-valued transformation from label matrix **Y**
  - SSDH avoids the direct optimization on $n \times n$ large matrix **S**, which makes the space cost acceptable when dealing with large-scale data.
  - Using real-valued matrix and hash codes rather than only binary matrix can permit more accurate approximation of **S**
- It leverages an alternating discrete optimization algorithm to efficiently learn the hash codes
  - scalable to deal with the large-scale datasets

Two step Hashing

- Divides the learning of hash codes and hash functions into two steps
  - First step: generation of hash codes by the supervision of loss functions
  - Second step: given the learnt hash codes, they learn hash functions which can transform the original features into the compact binary codes
- The performance of a two-step hashing method highly depends on the quality of the hash codes learnt in the first step
  - Two-step hashing focuses more on the criterion of generating hash codes, i.e. the design of loss functions
  - Once the hash codes are learnt, for any bit of the hash codes, learning the corresponding hash function to project features into it can be modelled as a binary classification problem
    - linear classifiers, SVM with RBF kernel, CNN
- SSDH belongs to two-step hashing
  - It’s novelty comes from it’s loss function and the corresponding optimization algorithm

Notation and Problem Definition

- n labeled instances $\boldsymbol{x_i} \in R^d$ with its label vector $\boldsymbol{y_i} \in \{0,1\}^c$ (i=1,2…n).
- Feature matrix: $\mathbf{X} \in R^{n \times d}$
- Label matrix: $\mathbf{Y} \in \{0,1\}^{n \times c}$
  - $\boldsymbol{Y}_{ik}$ is the k-th element of $\boldsymbol{y}_i^T$. $\boldsymbol{Y}_{ik} = 1$, if x_i belongs to class k, and $\boldsymbol{Y}_{ik} = 0$ otherwise.
- Instance-pairwise semantic similarity $\mathbf{S} \in \{-1,1\}^{n \times n}$
- SSDH aims to learn a *r*-bit binary hash code $b_i \in \{-1,1\}^r$ for each instance.
  - $\mathbf{B} \in \{-1,1\}^{n \times r}$ is the hash code matrix.
- Hash function F transforms **X** to **B**, i.e. **B**=F(**X**) =*sgn*(**XW**).
  - **W** is the projection matrix
  - *sgn*(·) is an element-wise sign function defined as *sgn*(*x*) = 1 if $x \geq 0$, and − 1 otherwise
- Without loss of generality, we assume the input instances to be zero centered, i.e. $\sum_{i=1}^{n} \boldsymbol{x}_i = 0$

Training of SSDH

- Two step Hashing
  - Hash codes learning step
  - Hash function learning step

Hash codes Learning-Design of Loss Function

- To leverage the semantic information to learn the hash codes, many hashing methods embed the similarity matrix into the following loss function

$$\min_{\boldsymbol{B}} \|r\boldsymbol{S} - \boldsymbol{B}\boldsymbol{B}^T\|_F^2, \mathrm{s.t.}\, \mathbf{B} \in \{-1,1\}^{n\times r} \quad (1)$$

The inner product of hash codes reflects the opposite of the Hamming distance. Eq. (1) uses the inner product to approximate the semantic similarities with the square loss.

  - The main drawback of this model is that it is time-consuming for optimization when *n* is large

- Proposed novel loss function

$$\min_{\boldsymbol{B},\boldsymbol{G}} \|r\boldsymbol{S} - \boldsymbol{B}(\boldsymbol{Y}\boldsymbol{G})^T\|_F^2 + \mu\|\boldsymbol{B} - \boldsymbol{Y}\boldsymbol{G}\|_F^2, s.t. \mathbf{B} \in \{-1,1\}^{n\times r} \quad (2)$$

  - $\boldsymbol{G} \in R^{c\times r}$ is a projection from **Y** to **B**, **Y** is the label matrix.

Hash codes Learning-Design of Loss Function

- Advantages of Eq. (2)
  - The ability to supervise the learning of hash codes by leveraging both pairwise similarity **S** and label matrix **Y** rather than only one of them, which can ensure more precise hash codes
  - The use of real-valued **YG** to replace one binary matrix **B** in Eq. (1)
    - Using the real-valued information rather than the binary **B** can obtain more accurate approximation of similarity **S**.
    - **YG** has been proved effective to approximate **B**.
  - By introducing **YG** and replacing one **B**, they can tactfully avoid the direct optimization on large **S**
    - They can compute **SY** offline and directly use the term of **SY** instead of **S** in the learning.
    - The size of **SY** is only $n \times c$ (c is usually much smaller than n)
      - solving the optimization problem can become much more efficient

Hash codes Learning-Optimization of Loss Function

- Alternating Strategy
  - Updating **G** by fixing **B**
  - Updating **B** by fixing **G**
    - Single-Label Data
    - Multi-Label Data

Updating **G** by fixing **B**

- Closed form solution of Eq. (2):

$$\boldsymbol{G} = (\boldsymbol{Y}\boldsymbol{Y}^T)^{-1}(r(\boldsymbol{S}\boldsymbol{Y})^T\boldsymbol{B} + \mu\mathbf{Y}^T\boldsymbol{B})(\boldsymbol{B}^T\boldsymbol{B} + \mu\boldsymbol{I}_{r\times r})^{-1} \quad (3)$$

- But if we use the matrix **S** the space and time cost of the optimization is unacceptable.
  - They can compute **SY** offline and load it directly during optimization.
- Denoting **A**=**SY**,

$$\boldsymbol{G} = (\boldsymbol{Y}\boldsymbol{Y}^T)^{-1}(r(\boldsymbol{A})^T\boldsymbol{B} + \mu\mathbf{Y}^T\boldsymbol{B})(\boldsymbol{B}^T\boldsymbol{B} + \mu\boldsymbol{I}_{r\times r})^{-1} \quad (4),$$

- **A** is $n \times c$ and c is much smaller than n.
  - So during optimization, SSDH can bypass the drawback of using large pairwise similarity matrix.

Updating **B** by fixing **G** – Single Label Data

- For the first term in Eq. (2) they replace the Frobenius Norm with the L1 norm, so we get

$$\min_{\boldsymbol{B}} \|r\boldsymbol{S} - \boldsymbol{B}(\boldsymbol{YG})^T\|_1, \text{s.t.}\, \mathbf{B} \in \{-1,1\}^{n\times r} \quad (5)$$

- The solution is **B**=*sgn*(**SYG**).
- For single label data *sgn*(**SYG**)= *sgn*(**YG**).
  - For the first term in Eq. (2) **B**=*sgn*(**YG**) is the solution.
  - The solution for the second term in Eq. (2) is also **B**=*sgn*(**YG**)
- So for single label data the solution to Eq. (2) is

B=*sgn*(**YG**) (6)

Updating **B** by fixing **G** – Multi Label Data

- Now *sgn*(**SYG**)≠ *sgn*(**YG**) and **B**=*sgn*(**YG**) is no longer a solution to Eq. (2).
- Discrete cyclic coordinate descent (DCC) is used to obtain **B** with a closed form solution. Omitting the constant terms $\|r\boldsymbol{S}\|_F^2, \mu\|\boldsymbol{B}\|_F^2, \mu\|\boldsymbol{YG}\|_F^2$, Eq. (2) is rewritten as

$$\min_{\boldsymbol{B}} \|\boldsymbol{YGB}^T\|_F^2 - 2Tr(\boldsymbol{B}^T\boldsymbol{SYG} + \mu\boldsymbol{B}^T\boldsymbol{YG})$$
$$= \|\boldsymbol{YGB}^T\|_F^2 - 2Tr\left(\boldsymbol{B}^T(\boldsymbol{AG} + \mu\boldsymbol{YG})\right)$$
$$= \|\boldsymbol{CB}^T\|_F^2 - 2Tr(\boldsymbol{B}^T\boldsymbol{Q}) \text{ s.t. } \mathbf{B} \in \{-1,1\}^{n\times r},$$

where **A**=**SY**, **C**=**YG** and **Q**=**AG**+μ**C**

- Then they solve **B** bit by bit
  - they iteratively learn every column of **B** by fixing all other columns.
- Suppose **b** is the l-th column of **B**, i.e. **b** is the vector composed of the l-th bits of all samples, l=1,…r and **B´** is the matrix excluding **b**. **q** is the l-th row of **Q**, **Q´** is the matrix **Q** excluding **q** and **c** is the l-th row of **C**, **C´** the matrix **C** excluding **c**.

Updating **B** by fixing **G** – Multi Label Data

- Then we have

$$\|\boldsymbol{CB}^T\|_F^2 = Tr(\boldsymbol{BC}^T\boldsymbol{CB}^T)$$
$$= \|\boldsymbol{bc}\|_F^2 + 2\boldsymbol{cC}'\boldsymbol{B}'^T\boldsymbol{b} + const$$
$$= 2\boldsymbol{c}^T\boldsymbol{C}'\boldsymbol{B}'^T\boldsymbol{b} + const$$

- Here $\|\boldsymbol{bc}\|_F^2 = Tr(\boldsymbol{c}^T\boldsymbol{b}^T\boldsymbol{bc}) = n\boldsymbol{c}^T\boldsymbol{c} = const$. Correspondingly we have

$$2Tr(\boldsymbol{B}^T\boldsymbol{Q}) = \boldsymbol{q}^T\boldsymbol{b} + const.$$

- Then the optimization problem can be reduced to the following form

$$\min_{\boldsymbol{c}}(\boldsymbol{c}^T\boldsymbol{C}'\boldsymbol{B}'^T + \boldsymbol{q}^T)\boldsymbol{b} \text{ s.t. } \mathbf{b} \in \{-1,1\}^n.$$

- And the optimal solution is

$$\boldsymbol{b} = sgn(\boldsymbol{q} - \boldsymbol{B}'\boldsymbol{C}'^T\boldsymbol{c}) \quad (11)$$

- Each bit b is computed based on the pre-learnt bits of **B**. We can iteratively update each bit until the procedure converges to a set of better codes B.

The Whole Algorithm

Algorithm 1: Discrete optimization in SSDH.

Input: $\mathbf{A} = \mathbf{SY}$, which is computed offline; the label matrix $\mathbf{Y} \in \{0, 1\}^{n \times c}$; the parameter μ; the iteration number t; the hash bit length r.

Output: Hash codes $\mathbf{B} \in \{-1, 1\}^{n \times r}$; the projection matrix $\mathbf{G}$.

Initialize $\mathbf{G}$ and $\mathbf{B}$ by randomization.

for $iter = 1 \rightarrow t$ **do**

  Update $\mathbf{G}$ by fixing $\mathbf{B}$, using Eq. (4).

  if *single-label* **then**

    Update $\mathbf{B}$ by fixing $\mathbf{G}$, using Eq. (6).

  end

  if *multi-label* **then**

    Update $\mathbf{B}$ by fixing $\mathbf{G}$, using Eq. (11).

  end

end

Hash Function Learning

- Once the hash codes are learnt, SSDH needs to learn hash function which can transform the original features into binary codes
- SSDH uses linear regression (as hash function)
- The hash function is obtained solving

$$L_e = \|\boldsymbol{B} - \boldsymbol{X}\boldsymbol{W}\|_F^2 + \lambda_e \|\boldsymbol{W}\|_F^2,$$

where λ_e balance parameter and $\|\boldsymbol{W}\|_F^2$ a regularization term.

- The optimal solution is

$$\boldsymbol{W} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda_e \boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{B}.$$

Out-of-Sample Extension

$$\boldsymbol{B}_{query} = F\left(\boldsymbol{X}_{query}\right) = sgn(\boldsymbol{X}_{query}\boldsymbol{W})$$

Kernelization

- For data point $\boldsymbol{x} \in R^d$ they randomly sampled m anchor points i.e. $\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots \boldsymbol{o}_m$ and map $\mathbf{x}$ into an m-dimensional kernel feature representation using

$$\varphi(x) = \left[\exp\left(\frac{\|\boldsymbol{x}-\boldsymbol{o}_1\|^2}{\sigma}\right), \ldots, \exp\left(\frac{\|\boldsymbol{x}-\boldsymbol{o}_m\|^2}{\sigma}\right)\right]$$

where σ is the kernel width.

- m=1000 and σ is estimated according to the average Euclidean distances between the training samples.
- It can help to better capture the underlying nonlinear structure from the original features

Experimental Results

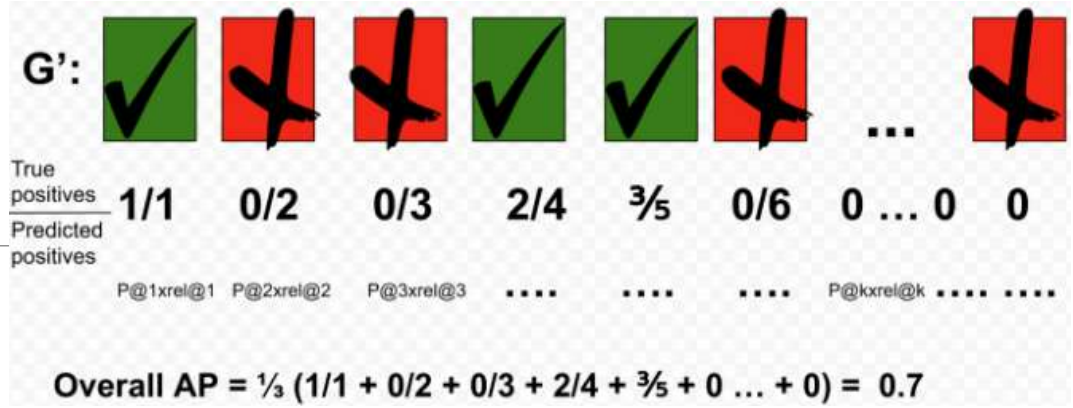
- Competing Methods:
  - *LSH*: Locality-Sensitive Hashing (unsupervised). Its basic idea is to hash the points from the database so as to ensure that the probability of collision is much higher for objects that are close to each other than for those that are far apart.
  - *ITQ*: Iterative Quantization (unsupervised). ITQ contains a simple and efficient alternating minimization scheme for finding a rotation of zero-centered data so as to minimize the quantization error of mapping this data to the vertices of a zero-centered binary hypercube
  - *KSH*: Kernel-Based Supervised Hashing (supervised). KSH utilizes the equivalence between optimizing the code inner products and the Hamming distances
  - *SDH*: Supervised Discrete Hashing (supervised). Its learning objective is to generate the optimal binary hash codes for linear classification
  - *NSH*: Natural Supervised Hashing (supervised). The key idea of NSH is to treat label vectors as binary codes and to learn target codes which have similar structure to label vectors
  - *FSDH*: Fast Supervised Discrete Hashing (supervised). FSDH uses a very simple yet effective regression of the class labels of training examples to the corresponding hash code to accelerate the algorithm
  - *COSDISH*: Column Sampling based Discrete Supervised Hashing (supervised). COSDISH directly learns the discrete hash code from semantic information
- For SSDH they used t=5, μ=1, λ_e=1.

General Statistics of Datasets

dataset	label kind	training size	query size	number of labels
MNIST	single-label	69,000	1,000	10
MIRFlickr	multi-label	15,738	1,000	24
CIFAR-10	single-label	59,000	1,000	10
CIFAR-100	single-label	59,000	1,000	100
NUS-WIDE	multi-label	193,833	2,100	21
ImageNet	single-label	1,198,336	63,070	1,000

- Each dataset is randomly split into a training set and a query set
- For single-label datasets, if two samples have the same class label, they are considered to be semantically similar, and dissimilar otherwise.
- For multi-label datasets, if two samples share at least one semantic label, they are considered to be semantically similar

Evaluation Metrics



- Average Precision: $AP = \frac{1}{R}\sum_{r=1}^{n} Precision(r)\delta(r)$
- For a query set of size M,

$$MAP = \frac{1}{M}\sum_{i=1}^{M} AP_i$$

- Precision-recall curves can be obtained by varying the Hamming radius of the retrieved points and evaluating the precision, recall and the number of retrieved points.
- TopN-precision curves reflects the change of precision with respect to the number of top-ranked *N* instances returned to the users,
  - expressive for retrieval

MAP results and training time on MNIST

	MAP					Training Time (in second)				
Method	8 bits	16 bits	32 bits	64 bits	96 bits	8 bits	16 bits	32 bits	64 bits	96 bits
LSH	0.1731	0.2202	0.2686	0.3267	0.3553	0.040	0.044	0.062	0.091	0.129
ITQ	0.3818	0.4132	0.4367	0.4601	0.4673	0.892	1.42	2.72	5.27	7.18
KSH	0.7103	0.7880	0.8258	0.8434	0.8428	48.02	90.72	179.82	337.53	508.68
SDH	0.5956	0.8544	0.8861	0.8945	0.8963	6.45	7.13	17.27	59.60	127.06
NSH	0.7314	0.8667	0.8934	0.9102	0.9105	4.33	5.25	5.38	6.89	8.51
FSDH	0.8701	0.8940	0.9139	0.9200	0.9253	8.60	7.93	8.71	9.94	10.46
COSDISH	0.7895	0.8498	0.8740	0.8866	0.8861	4.59	8.08	30.31	107.01	241.08
SSDH	**0.9338**	**0.9557**	**0.9639**	**0.9669**	**0.9683**	3.68	3.80	3.81	3.93	4.09

MAP results and training time on MIRFlickr

	MAP					Training Time (in second)				
Method	8 bits	16 bits	32 bits	64 bits	96 bits	8 bits	16 bits	32 bits	64 bits	96 bits
LSH	0.5669	0.5760	0.5712	0.5862	0.5885	0.005	0.005	0.007	0.013	0.016
ITQ	0.5760	0.5721	0.5794	0.5796	0.5806	0.134	0.154	0.305	0.589	0.925
KSH	0.5780	0.5775	0.5794	0.5782	0.5790	28.69	57.02	109.05	219.11	329.99
SDH	0.6023	0.6071	0.6138	0.6207	0.6220	1.31	1.90	3.99	13.19	30.38
NSH	0.6143	0.6180	0.6158	0.6243	0.6270	0.873	0.932	1.03	1.31	1.59
FSDH	0.5908	0.5957	0.6053	0.6124	0.6152	2.01	2.02	2.07	2.13	2.20
COSDISH	0.6819	0.6939	0.7036	0.7173	0.7185	0.639	1.66	6.04	29.85	69.99
SSDH	**0.7099**	**0.7128**	**0.7154**	**0.7198**	**0.7258**	0.722	0.849	1.32	3.10	6.29

MAP results and training time on CIFAR-10

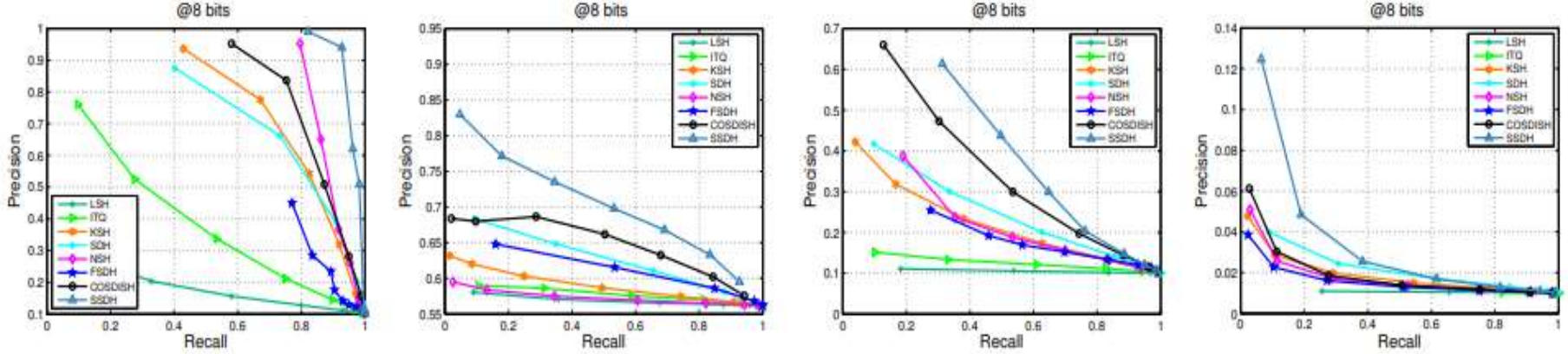
	MAP					Training Time (in second)				
Method	8 bits	16 bits	32 bits	64 bits	96 bits	8 bits	16 bits	32 bits	64 bits	96 bits
LSH	0.1077	0.1165	0.1199	0.1270	0.1360	0.021	0.025	0.032	0.044	0.064
ITQ	0.1429	0.1482	0.1569	0.1487	0.1551	0.483	0.743	1.32	2.54	3.98
KSH	0.2315	0.2623	0.2905	0.3095	0.3199	29.91	59.59	121.00	243.73	359.11
SDH	0.2373	0.3528	0.3868	0.4049	0.4159	4.58	5.55	9.09	29.54	83.13
NSH	0.2902	0.2936	0.3220	0.3390	0.3687	2.99	3.675	3.70	4.81	5.75
FSDH	0.3311	0.3884	0.4270	0.4522	0.4646	6.05	6.09	6.26	6.65	7.05
COSDISH	0.5012	0.5745	0.6094	0.6321	0.6394	2.10	4.33	15.19	76.99	180.34
SSDH	**0.5151**	**0.6698**	**0.6991**	**0.7046**	**0.7063**	2.61	2.69	2.66	2.67	2.73

MAP results and training time on CIFAR-100

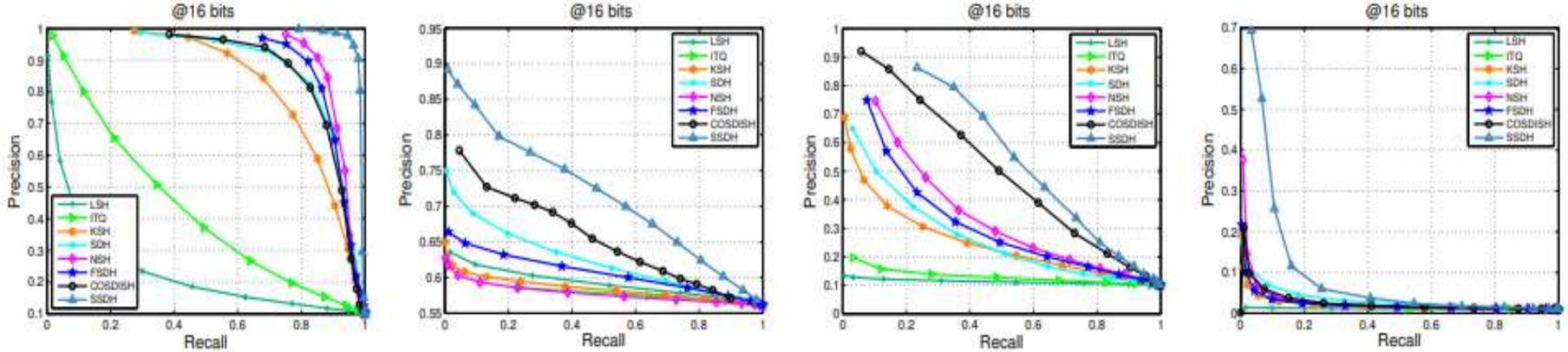
	MAP					Training Time (in second)				
Method	8 bits	16 bits	32 bits	64 bits	96 bits	8 bits	16 bits	32 bits	64 bits	96 bits
LSH	0.0117	0.0125	0.0139	0.0153	0.0169	0.021	0.025	0.027	0.0485	0.059
ITQ	0.0125	0.0151	0.0170	0.0193	0.0203	0.468	0.711	1.28	2.46	4.01
KSH	0.0225	0.0268	0.0290	0.0307	0.0338	32.63	63.60	128.91	257.12	386.89
SDH	0.0217	0.0339	0.0473	0.0589	0.0392	3.83	5.77	9.69	80.92	190.57
NSH	0.0323	0.0483	0.0649	0.0758	0.0837	3.24	3.53	4.07	5.58	6.99
FSDH	0.0256	0.0419	0.0598	0.0690	0.0884	6.61	6.55	6.73	7.01	7.57
COSDISH	0.0379	0.0752	0.1449	0.2019	0.2292	1.99	6.39	16.76	79.82	185.28
SSDH	**0.1001**	**0.1640**	**0.2348**	**0.2824**	**0.2674**	3.03	3.11	3.16	3.30	3.34

- From the MAP and time cost results in these table, they conclude SSDH is effective and efficient on these datasets

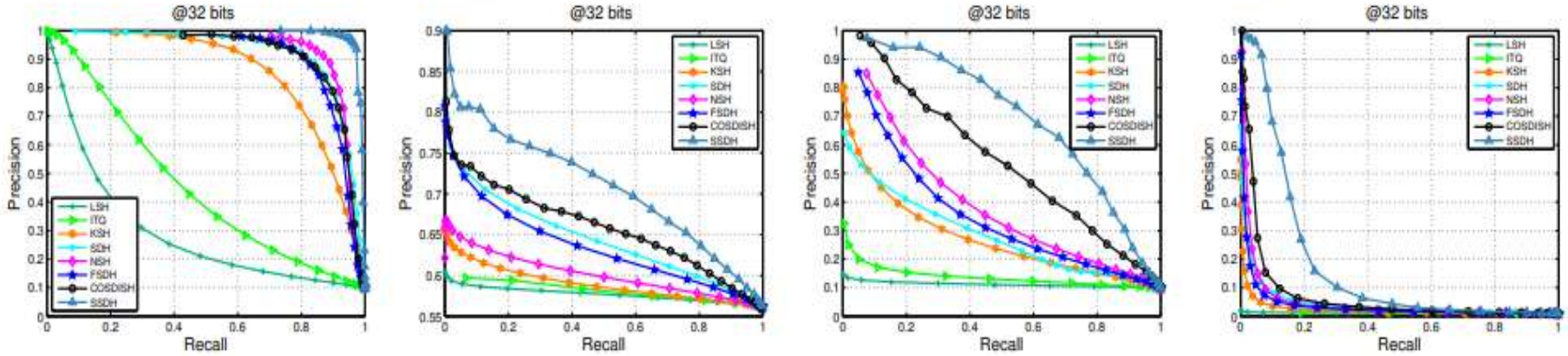
Precision-Recall curves



(a) MNIST @8 bits (f) MIRFlickr @8 bits (k) CIFAR-10 @8 bits (p) CIFAR-100 @8 bits

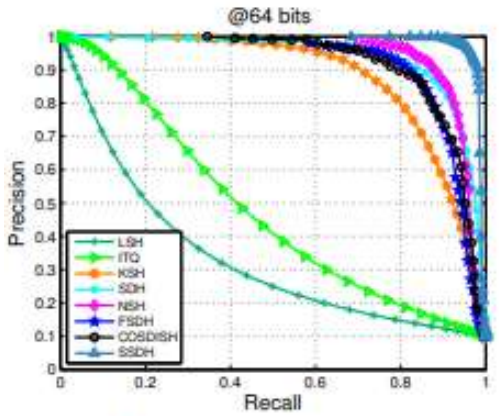


(b) MNIST @16 bits (g) MIRFlickr @16 bits (l) CIFAR-10 @16 bits (q) CIFAR-100 @16 bits

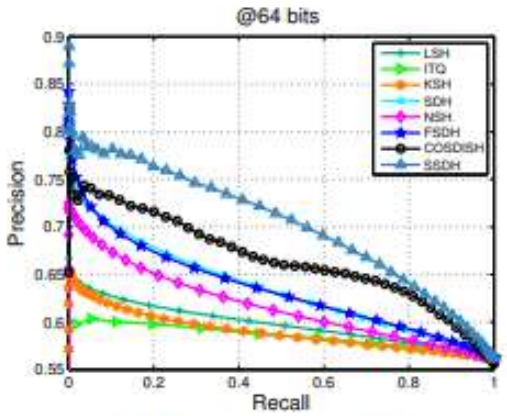


(c) MNIST @32 bits (h) MIRFlickr @32 bits (m) CIFAR-10 @32 bits (r) CIFAR-100 @32 bits

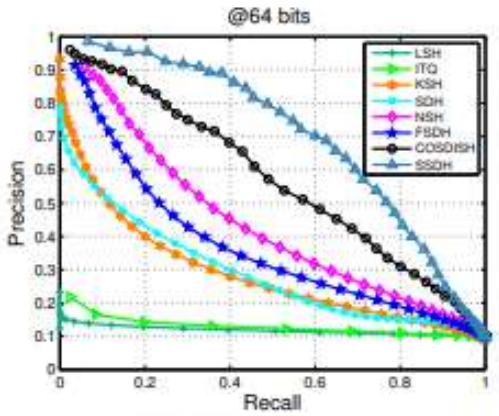
Precision-Recall curves



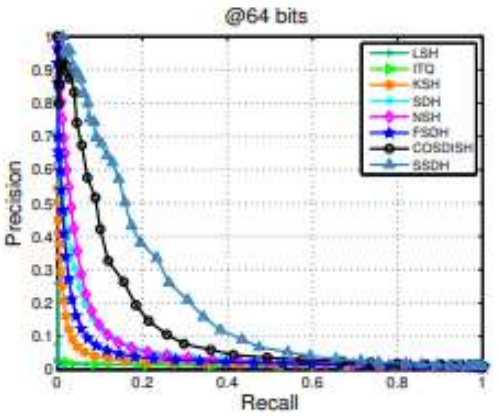
(d) MNIST @64 bits



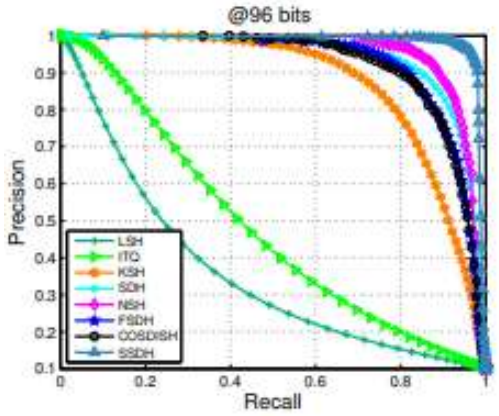
(i) MIRFlickr @64 bits



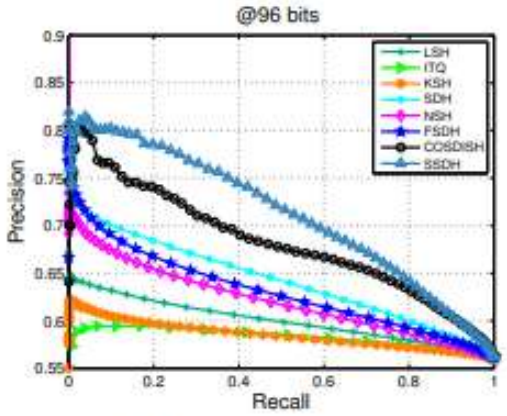
(n) CIFAR-10 @64 bits



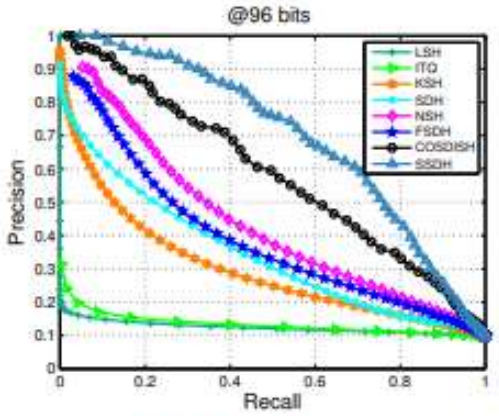
(s) CIFAR-100 @64 bits



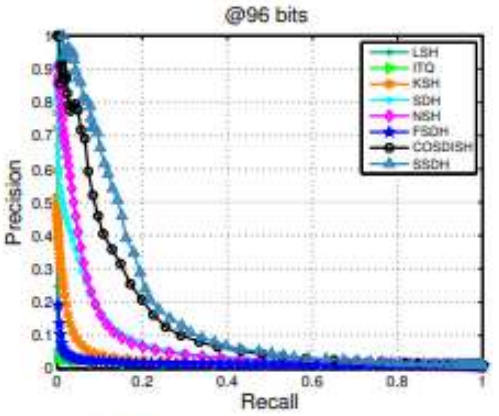
(e) MNIST @96 bits



(j) MIRFlickr @96 bits

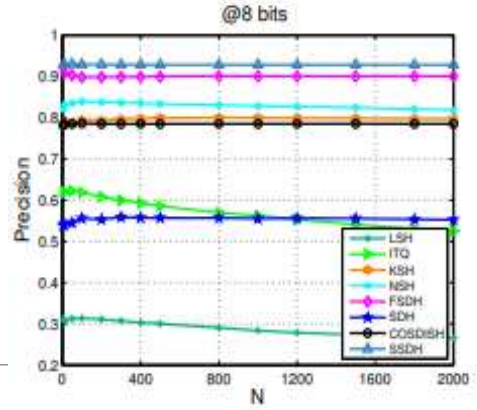


(o) CIFAR-10 @96 bits

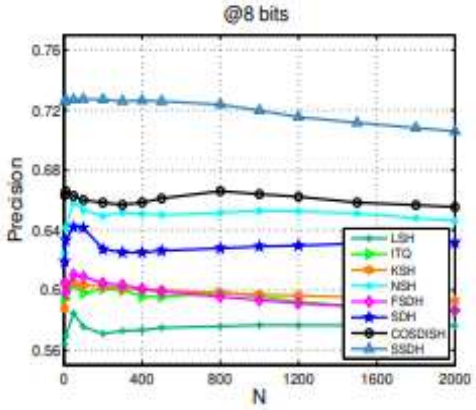


(t) CIFAR-100 @96 bits

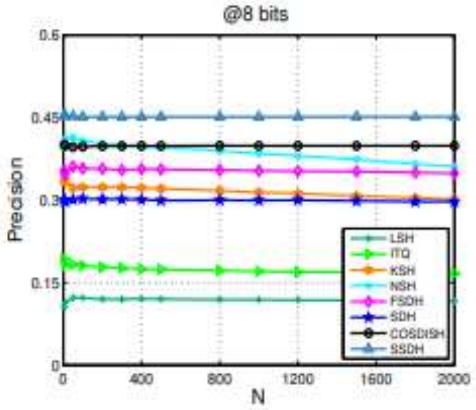
TopN-Precision curves



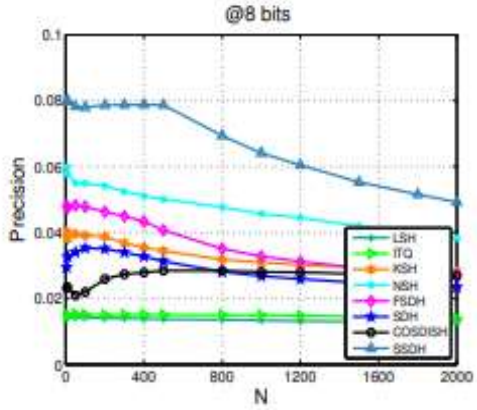
(a) MNIST @8 bits



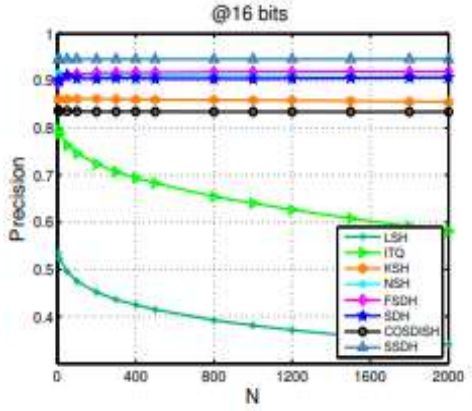
(f) MIRFlickr @8 bits



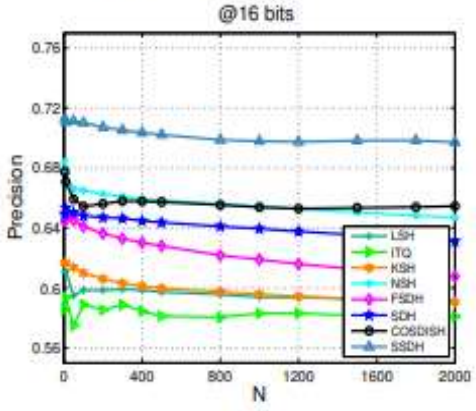
(k) CIFAR-10 @8 bits



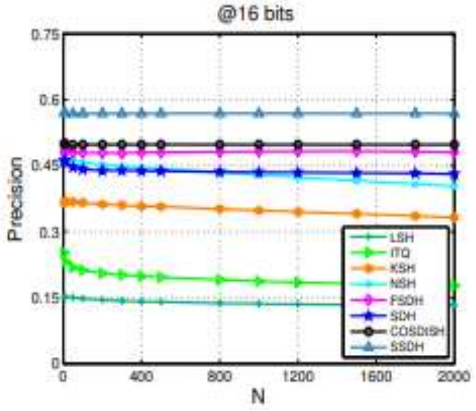
(p) CIFAR-100 @8 bits



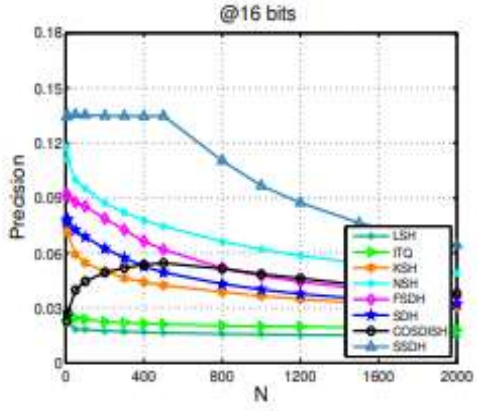
(b) MNIST @16 bits



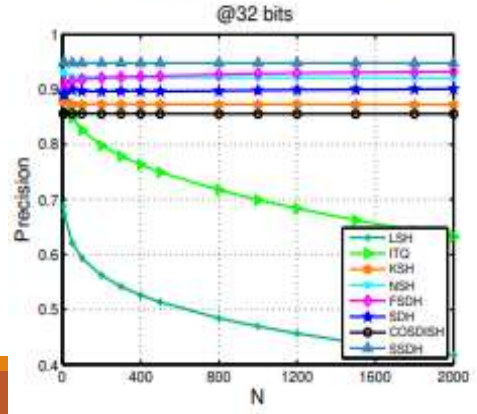
(g) MIRFlickr @16 bits



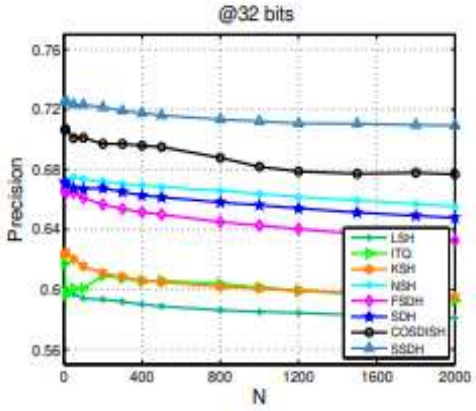
(l) CIFAR-10 @16 bits



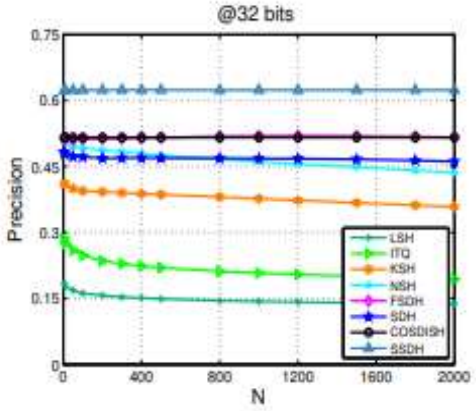
(q) CIFAR-100 @16 bits



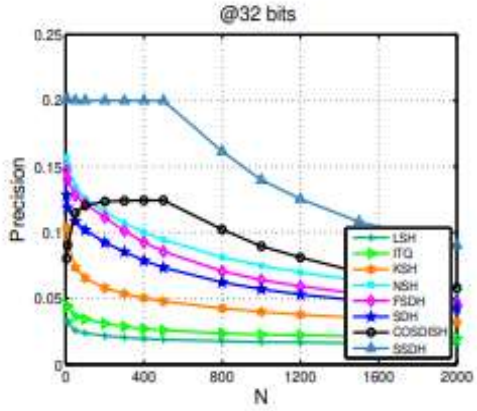
(c) MNIST @32 bits



(h) MIRFlickr @32 bits

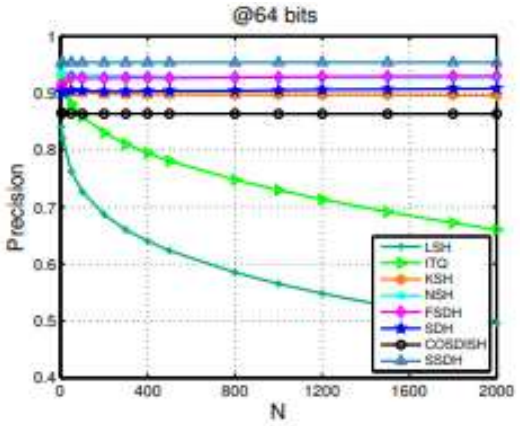


(m) CIFAR-10 @32 bits

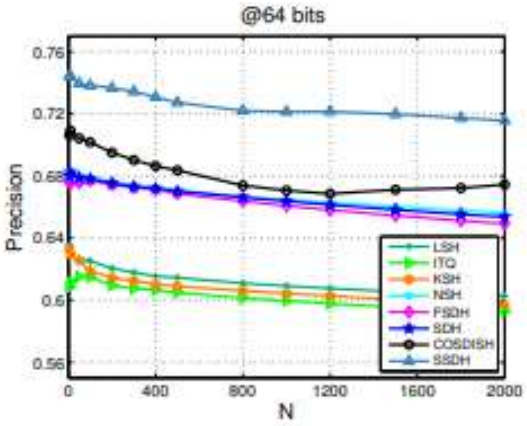


(r) CIFAR-100 @32 bits

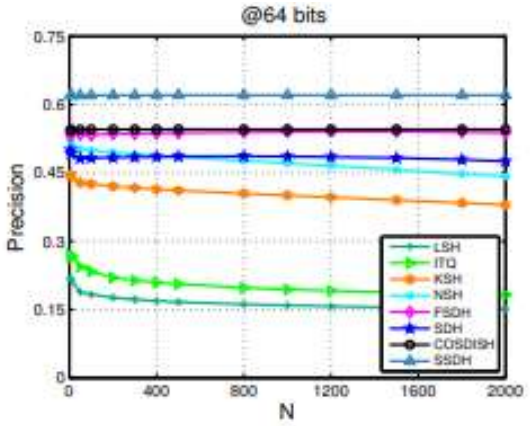
TopN-Precision curves



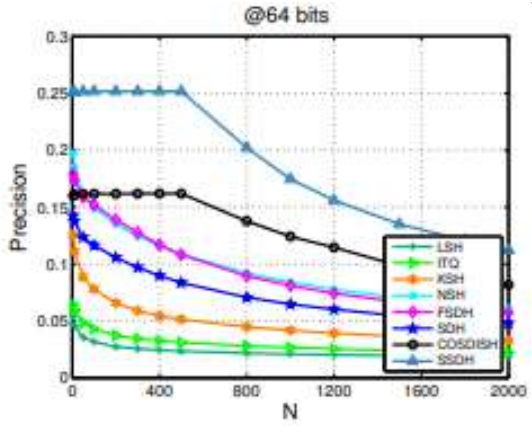
(d) MNIST @64 bits



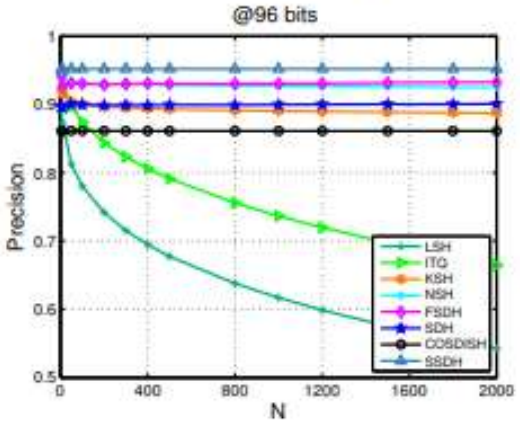
(i) MIRFlickr @64 bits



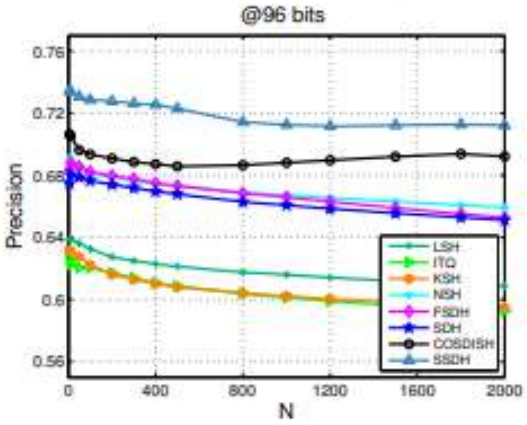
(n) CIFAR-10 @64 bits



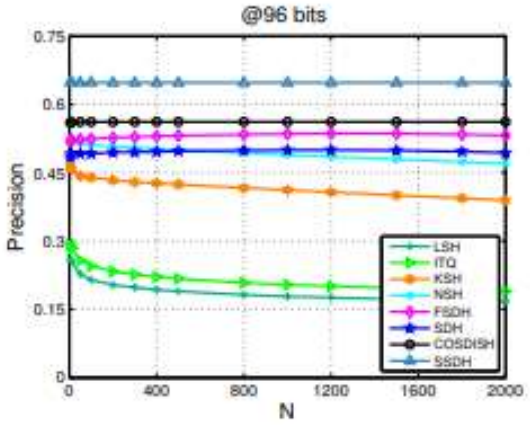
(s) CIFAR-100 @64 bits



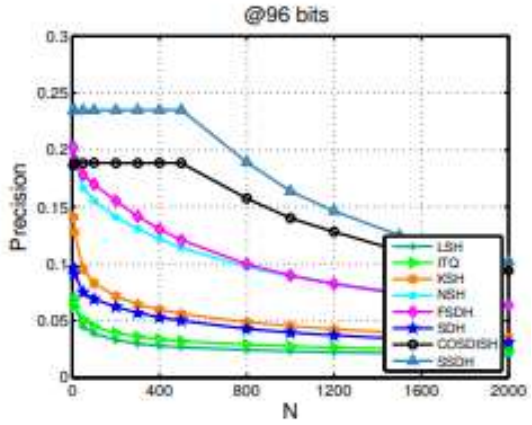
(e) MNIST @96 bits



(j) MIRFlickr @96 bits



(o) CIFAR-10 @96 bits



(t) CIFAR-100 @96 bits

MAP results and training time on NUS-WIDE

	MAP			Training Time (in second)		
Method	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
LSH	0.3107	0.3165	0.3197	0.06	0.08	0.13
ITQ	0.3343	0.3348	0.3364	2.55	5.53	9.63
KSH	0.3703	0.3728	0.3785	52.53	105.24	208.97
SDH	0.4113	0.4114	0.4135	28.94	77.95	250.95
NSH	0.4075	0.4135	0.4133	10.97	11.93	15.26
FSDH	0.4052	0.4115	0.4155	19.86	19.26	21.48
COSDISH	0.5765	0.5947	0.5942	16.11	56.27	232.55
SSDH	**0.6023**	**0.6035**	**0.6069**	10.88	20.86	56.02

- In a nutshell, SSDH is the most practical supervised hashing method, especially for large-scale datasets

MAP results and training time on ImageNet

	MAP			Training Time (in second)		
Method	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
LSH	0.0015	0.0018	0.0021	0.40	0.51	0.92
ITQ	0.0027	0.0028	0.0031	23.16	35.92	60.20
KSH	0.0023	0.0020	0.0024	41.81	92.13	228.22
SDH	0.0026	0.0038	0.0051	979.42	1859.35	5327.54
NSH	0.0045	0.0052	0.0065	92.46	112.55	145.41
FSDH	0.0027	0.0037	0.0051	222.28	230.02	248.99
COSDISH	0.0041	0.0141	0.0212	99.33	344.98	1837.24
SSDH	**0.0122**	**0.0209**	**0.0335**	269.38	282.78	290.51

- From the results on ImageNet, they further conclude that SSDH scales well on large-scale dataset (more than 1 million) and is the most practical supervised hashing method.

Weaknesses of SSDH

- The reduced space cost of similarity matrix is still large and relevant to the amount of data n.
- SSDH adopts the bit-wise learning strategy to generate binary codes, making it time-consuming.
- SSDH only leverages the semantic information to learn the hash codes, making it not robust to noise
  - fails to exploit the inherent features in the training data

Fast Scalable Supervised Hashing

- FSSH is more scalable to large datasets
  - By embedding the large similarity matrix into an intermediate term whose size is independent of the amount of data n
- Instead of learning the hash codes bit by bit, it learns all bits simultaneously making the training time of FSSH robust to the hash code lengths
- FSSH can jointly harness the relations in the data and the semantic information

FSSH Loss Function

$$\min_{\boldsymbol{B},\boldsymbol{G},\boldsymbol{W}} \|\boldsymbol{S} - \phi(\boldsymbol{X})\boldsymbol{W}(\boldsymbol{L}\boldsymbol{G})^T\|_F^2 + \mu\|\boldsymbol{B} - \boldsymbol{L}\boldsymbol{G}\|_F^2 + \theta\|\boldsymbol{B} - \phi(\boldsymbol{X})\boldsymbol{W}\|_F^2$$
$$s.t. \mathbf{B} \in \{-1,1\}^{n\times r}$$

Method	CIFAR-10			
	16 bits	32 bits	64 bits	96 bits
SSDH	**0.6698**	**0.6991**	0.7046	0.7063
FSSH_ts	0.6350	0.6829	**0.7071**	**0.7108**

Method	MNIST			
	16 bits	32 bits	64 bits	96 bits
SSDH	0.9557	0.9639	0.9669	0.9683
FSSH_ts	**0.9443**	**0.9649**	**0.9713**	**0.9721**

Method	NUS-WIDE			
	16 bits	32 bits	64 bits	96 bits
SSDH	**0.6023**	0.6035	0.6069	0.6101
FSSH_ts	0.5846	**0.6060**	**0.6105**	**0.6225**

Method	NUS-WIDE (193K)			
	16 bits	32 bits	64 bits	96 bits
SSDH	14.47	25.60	61.54	129.84
FSSH_ts	13.71	14.09	15.51	15.53

Thank You!