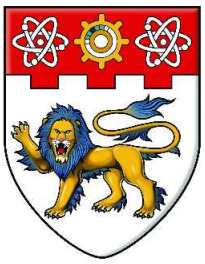


NANYANG
TECHNOLOGICAL
UNIVERSITY

Deep Learning with Application to Hashing

A thesis submitted
for a degree of Master of Engineering
by

ZHANG Boshen

School of Computer Engineering
Nanyang Technological University

July 2013

Acknowledgments

I would express my deep gratitude to my main supervisor Dr. Xiaokui Xiao who provides me funding on my research. I can't get my degree without his help. I would also express my deep gratitude to my co-supervisor Dr. Steven HOI, who brought me into the beautiful world of machine learning.

I would thank Dr.Peilin Zhao from NTU And Mr Nitish Srivastava, PhD student from University of Toronto, who gave me a lot of help when I met problems in deep learning.

I would also thank all my friends in DISCO lab, NTU.

Last but not least, I would thank Mr and Mrs Zhang, who gave me life, and have been supporting me in the past 25 years. I will never forget that.

Contents

List of Figures

List of Tables

Abstract

Deep Learning and Learning to Hash are two important research areas in machine learning, which have rapid improvements in recent years.

What I mainly researched on is an inter-discipline field: deep learning for cross view hashing. Multiple layers of representation in deep learning has the property of abstracting representation from input data, while, in the cross view similarity search, the biggest difficulty is to represent items from one domain to another. Here, I want to take advantage of the latest deep learning technology to solve the cross view similarity search problem. Hashing is used to accelerate this process.

This thesis mainly contains three parts. Chapter 2 is a literature survey. It contains a deep learning survey and a learning to hash survey. The deep learning survey briefly introduces fundamental technology of deep learning and its recent development including the latest technology. The Learning to Hash survey brief introduces some widely used learning to hash algorithms. Chapter 3 is an experiment about comparison of some state of the arts learning to hash algorithms. Chapter 4 is cross view hashing based on deep learning. I present a cross view feature hashing technique using deep learning and show some results. These three chapters are main chapters. Chapter 1 and Chapter 5 are introduction and conclusion.

Chapter 1

Introduction

This introduction will present a brief review of deep learning and hashing algorithms. Those concepts introduced here will be explained with more details in later chapters. My research involves mainly two fields in machine learning, including deep learning and learning to hash.

1.1 Deep Learning

As an emerging and active topic in machine learning community [Ben09a], deep learning is about learning multiple levels of representation and abstraction to help understand the data and resolve a variety of subsequent machine learning tasks. One challenging task of deep learning is to search for good parameters in the parameter space. Researchers have investigated some effective ways to attack this challenge by exploring learning algorithms such as Deep Belief Networks [HOT06] and variants based on the Restricted Boltzmann Machines (RBMs) [RB08, SH10, VLBM08, LGRN09].

1.2 Learning to Hash

In literature, (approximate) nearest neighbor search on single-view high-dimensional data has been extensively studied for many years in database and data mining communities. One recent promising technique is the family of learning to hash algorithms [IM98, DIIM04, KG09, TFW08, SH09b, WTF08, RL09, LCL12, LWKC11, NF11, GL11, KD, KL12, ZWCL10, WKC12, WKC10b, LWJ+12], which basically aim to map query and target instances in the same input data space (which can be of arbitrarily high dimensionality) to a low-dimensional binary (Hamming) space; As a result, efficient linear scans on the binary data can be trivially implemented to find the exact nearest neighbors in the Hamming space, or one can perform fast approximate nearest neighbor search on the Hamming space with running time that is sub-linear in the number of total instances.

1.3 Cross View Hashing with Deep Learning

Unlike the above regular nearest neighbor search on single-view data, the key challenge in the cross-view nearest neighbor search task is that the input spaces (views) of the query instance and the target instances to be retrieved can be diverse or completely different, which prevents the straightforward application of the existing learning to hash algorithms to solve the Cross-View Nearest Neighbor Search(CVNNS) tasks. To attack this challenge, one direction is to develop some technique that can directly map data instances from different input spaces to the same binary (Hamming) space. This approach usually cannot take advantages of many successful learning to hash algorithms for single-view data. Unlike this approach, I want to present a new deep learning to hash scheme for CVNNS, which explores the emerging deep learning techniques in learning a shared representation by mining training data to bridge the gap between the query view and the target view, and is able to take advantages of the existing learning to hash techniques for performing hashing on the shared representation.

1.4 Thesis Organization

Although this thesis focuses on cross view hashing with deep learning, I focus more on the applications of deep learning in my master period. Cross view hashing is only a tip of the iceberg.

The following chapters are organized as follows. Chapter 2 contains the literature survey of deep learning and learning to hash.Chapter 3 compares with the most widely used learning to hash algorithms. Chapter 4 introduces our cross view hashing framework based on deep learning. Chapter 5 contains conclusion and future work. I will give some ideas which may use deep learning technique to solve some existing problems in Chapter 5. Currently I do not have time to implement those ideas. If I pursue a PhD degree in the future, I will continue my research.

Chapter 2

Literature Survey of Deep Learning and Learning to Hash

2.1 Deep Learning Survey

Deep learning is a new Machine Learning area, which has been introduced with moving Machine Learning closer to its original goals: Artificial Intelligence [DLN]

2.1.1 What Is Deep Learning

Generally speaking, deep learning is about learning multiple level of representation and abstraction that help to analysis data such as sound, video, images, text and so on. It tries to model high order constrains between inputs and tries to find the internal factors among inputs [HS86, DLN, Ben09a, AHS87].

In deep architecture, representation of data is a layer-wise form. 'Deep learning' means the layers of the neural network are more than 2. [Ben09a] provides theoretical limitations of 'Shallow learning'. Deep models are more powerful. It is proved that functions that can be compactly represented by a depth k might require an exponential number of computational elements to be represented by a $k-1$ architecture.

The multiple-layer, hierarchy architecture is also an inspiration from biology and human cognition. It is shown that human brain is connected by synaptic and neurons [Eli05], which forms a deep network layer by layer, where one unit in deep network can represent the neuron and each edge connected units represent the synaptic. The weights of the edge represent the relationship between two neurons it connected or the strength between them. When information is transferred, the artificial neural network is used to simulate the information transformation in human brain.

2.1.2 Why It Is So Difficult To Train Deep Models

Looking back at the history of multi-layer neural networks, their problematic non-convex optimization has for a long time prevented reaping the expected benefits [Ben09b] of going beyond one or two hidden layers. There was no clear understanding of what constitutes good representations for initializing deep architectures or what explicit unsupervised criteria may best guide the learning. Standard gradient descent from random initialization is doing very poorly[HOT06, BG10], [Ben09b] shows that logistic sigmoid activation is unsuited for deep models with random initialization because of its own characteristic. [EMB+09] also shows that Pre-training, instead of random initialization, is a key part when training neural network.

After pre-training, one may rely on labelled data for fine-tuning. However, labelled data is often scarce. So it is difficult to get enough examples to fit parameters in some complex problems.

Another issue is optimization with local optima[HS86]. Training deep neural network involves solving a highly non-convex problem[DLS] with bad local optima, and training with gradient descent no longer work well. [DLS] also mentions a 'diffusion of gradients' which means gradient descent with back propagation, propagated gradients are diminished rapidly in magnitude as the depth of the network increase and cause the earlier layers fail to learn much.

2.1.3 Why Pre-training Helps Deep Learning

In deep learning, general there is a two-step training algorithm when training deep models. First step is pre-training. Second step is fine-tuning [HOT06]. As the layer of models and the number of samples increases, the number of parameters that we need to find maybe very huge. Tuning those parameters and find the best values among them is very difficult. The two-step training algorithm has the following meaning. At first pre-training step, we find a good initialization for the model, since random initialization has many drawbacks [EMB+09]. When pre-training is done, the parameter search range is reduced to a small playground. Then, in the second fine-tuning step, we use some other information(supervised or unsupervised information) to find a good local minimal in the small playground. This two-step algorithm usually works well for most of the deep models.

[EBC+10] shows that pre-training step works like a regularizer that initializes the parameters in a 'better' basin of attraction of the optimization procedure, corresponding to an apparent local minimum associated with better generalization.

2.1.4 Energy Based Models(EBM)

All models discussed below are energy based models(EBM). EBM associates a scalar energy to each configuration of the variables of interest. Learning corresponds to modifying that energy function so that its shape has desirable properties [DLN].

$$p(x) = \frac{e^{-E(x)}}{Z} \tag{2.1}$$

where the partition function is an analogy with physical systems

$$Z = \sum_{x} e^{-E(x)} \tag{2.2}$$

Then we can use the gradient descent on the negative log-likelihood of the training data. We define:

$$\begin{aligned} L(\theta, D) &= \frac{1}{N} \sum_{x^{(i)} \in D} \log p(x^{(i)}) \\ l(\theta, D) &= -L(\theta, D) \end{aligned} \tag{2.3}$$

Separating the example x into visible (v) and hidden(h), we then rewrite:

$$P(x) = \sum_{h} P(v, h) = \sum_{h} \frac{\exp^{(-E(v,h))}}{Z} \tag{2.4}$$

then define free energy as follows:

$$\begin{aligned} F(v) &= -\log \sum_{h} e^{-E(v,h)} \\ P(v) &= \frac{e^{-F(v)}}{Z} \\ Z &= \sum_{v} e^{-F(v)} \end{aligned} \tag{2.5}$$

2.1.5 Deep Models

In feed forward neural network, information moves one direction, never goes back. The simplest feed forward neural network is perception. which can be future classified into single layer perception and multi layer perception. Basically, there are two types of stochastic feed-forward network. One is sigmoid belief network, another is noisy-OR network. Besides feed forward neural network, there are recurrent neural networks(RNN) where connections between units form a directed cycle, we will see some examples of RNN later.

2.1.5.1 Sigmoid Belief Networks

Belief networks, also known as 'Bayesian networks', are designed to represent a probability distribution over a set of attributes [Nea92].

In belief net, the probability of a state vector is only determined by the units that precedes it:

$$P_i = \Pi_j P(s_i = x_i | s_j = x_j, j < i) \tag{2.6}$$

Here, the order of units in the network is crucial, since it determines which conditional probabilities must be specified.

Sigmoid belief network is a variant of belief net that was designed in analogy with Boltzmann Machine. Units take value $0/1$ or $-1, +1$ is equivalent [Nea92]. We take $0/1$ units for example, the forward conditional probabilities for a sigmoid belief networks can be defined as :

$$P(s_i = x_i | s_j = x_j, j < i) = \delta(\sum_{j<i}(2s_i - 1)s_j w_{i,j}) \tag{2.7}$$

Then, the probability of a global state vector, $\hat{s}$, is defined of :

$$P(\hat{S} = \hat{s}) = \Pi_i P(s_i = x_i | s_j = x_j, j < i) = \Pi_i \delta(\sum_{j<i}(2s_i - 1)s_j w_{i,j}) \tag{2.8}$$

2.1.5.2 Noisy OR Network

"Noisy-OR" model views the units as $0/1$ valued OR-gates. There is a certain probability that even an input unit j has value 1, it will fail to force unit i, that it feeds into to go to 1, if we represent this probability as $q_{i,j}$, The forward conditional probability can be expressed as follows:

$$P(S_i = 1 | S_j = s_j : j < i) = 1 - \pi_{j<i, s_j=1} * q_{i,j} \tag{2.9}$$

2.1.5.3 Hopfield Network

The above two networks are types of stochastic feed-forward network. In feed forward network, information moves one direction, never goes back. But most of the network models are recurrent neural network, that means information can go back and form some loops ,such as Hopfield Network [Hop82] and Boltzmann Machine [AHS87].

Hopfield Network is the simplest Recurrent neural network. It is invented by John Hopfield [Hop82]. It serves as content-addressable memory systems with binary threshold notes, that means if the incoming value is larger than the threshold, the state of the node will be 1, otherwise, will be -1.

Hopfield net is also an energy based model. Its energy is defined as

$$E = -\frac{1}{2}\sum_{i,j} w_{i,j} s_i s_j + \sum_i \theta_i s_i \tag{2.10}$$

Training algorithms must insure that when units are randomly chosen to update, the energy E must decrease. Hopfield net can be trained using Hebbian Learning rule.

2.1.5.4 Hebbian Learning Rule

When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased. This is often paraphrased as 'Neurons that fire together wire together'. It is commonly referred to as Hebb's Law [Heb47, Heb47].

Hopfield net uses Hebbing learning rule to change the weights between connections. The Hebbian learning rule specifies how much the weight of the connection between two units should be increased or decreased in proportion to the product of their activation. It works well as long as all the input patterns are orthogonal or uncorrelated. The requirement of orthogonality places serious limitations on the Hebbian learning rule. A more powerful learning rule is the Delta Rule [TDR], which utilizes the discrepancy between the desired and actual output of each output unit to change the weights feeding into it.

2.1.5.5 Delta Rule

When we train the model, the system first uses the input to produce its own output and then compares this with the desired output[RHW88]. Then the weights are changed to reduce the difference. A general used rule for changing weights following presentation of input and output pair is given by [Sto86, RHW88]:

$$\Delta_p w_{i,j} = \eta(t_{pj} - o_{pj}) i_{pi} = \eta \delta_{pj} i_{pi} \tag{2.11}$$

where t_{pj} is the target for the j_{th} component of the output pattern, o_{pj} is the j_{th} element of the actual output pattern. i_{pi} is the value of the i_{th} element of the input. $\delta_{pj} = t_{pj} - o_{pj}$ is the discrepancy of target output and actual output.

More specific, let

$$E_p = \frac{1}{2}\sum_j (t_{pj} - o_{pj})^2 \tag{2.12}$$

be the measure of input output pattern p and let $E = \sum_p E_p$ be the overall measure of error.

$$\frac{\partial E_p}{\partial w_{i,j}} = \frac{\partial E_p}{\partial o_{p,j}} \frac{\partial o_{pj}}{\partial w_{j,i}} = (-\delta_{pj}) \frac{\partial o_{pj}}{\partial w_{j,i}} \tag{2.13}$$

If units are linear, let's say:

$$o_{pj} = \sum_i w_{ji} i_{pi} \tag{2.14}$$

Then we can get that:

$$(-\delta_{pj}) \frac{\partial o_{pj}}{\partial w_{j,i}} = (-\delta_{pj}) * i_{pi} \tag{2.15}$$

Then we get that:

$$-\frac{\partial E_p}{\partial w_{i,j}} = \delta_{pj} i_{pi} \tag{2.16}$$

This equation exactly shows how the delta rule comes from. It implements a gradient descent in E when the units are linear.

The drawback is that with hidden units, the error surface is not concave upwards, so may face the problem of getting stuck in local minima. [RHW88] show the generalized delta rule can be applied to arbitrary networks. First of all, assume the model is layered feedforward netowrk. Thus,

$$net_{pj} = \sum_i w_{ji} o_{pi} \tag{2.17}$$

is the total input to unit j under pattern p. o_{pi} is the output of its preceded units. Then, we can define the output of unit j is a function of its total input net_{pj}

$$o_{pj} = f_i(net_{pj}) \tag{2.18}$$

where f is differentiable. Then we get:

$$\frac{\partial E_p}{\partial w_{i,j}} = \frac{\partial E_p}{\partial net_{p,j}} \frac{\partial net_{p,j}}{\partial w_{i,j}} \tag{2.19}$$

$$\frac{\partial net_{p,j}}{\partial w_{i,j}} = \frac{\partial \sum_k w_{j,k} o_{p,k}}{\partial w_{j,i}} = o_{p.i} \tag{2.20}$$

then define:

$$\delta_{p,j} = -\frac{\partial E_p}{\partial net_{p,j}} \tag{2.21}$$

Then we get that:

$$-\frac{\partial E_p}{\partial w_{j,i}} = \delta_{p,j} o_{p,i} \tag{2.22}$$

We then see $\delta_{p,j}$:

$$\delta_{p,j} = -\frac{\partial E_p}{\partial net_{p,j}} = -\frac{\partial E_p}{\partial o_{p,j}} \frac{\partial o_{p,j}}{\partial net_{p,j}} = -\frac{\partial E_p}{\partial o_{p,j}} f_j'(net_{pj}) \tag{2.23}$$

$$\frac{\partial E_p}{\partial o_{p,j}} = -(t_{pj} - o_{pj}) \tag{2.24}$$

so we get :

$$\delta_{pj} = (t_{pj-} o_{pj}) f_j^{'}(net_{p,j}) \tag{2.25}$$

If u_j is not an output unit, then

$$\sum_k \frac{\partial E_p}{\partial net_{p,k}} \frac{\partial net_{p,k}}{\partial o_{p,j}} = \sum_k \frac{\partial E_p}{\partial net_{p,k}} \frac{\sum_i w_{ki} o_{pi}}{\partial o_{p,j}} = \sum_k \frac{\partial E_p}{\partial net_{p,k}} w_{k,j} = \sum_k \delta_{p,k} w_{k,j} \tag{2.26}$$

Then

$$\delta_{pj} = f_j^{'}(net_{p,j}) \sum_k \delta_{pk} w_{kj} \tag{2.27}$$

Specifically, if we use logistic function in which:

$$o_{p,j} = \frac{1}{1 + e^{-(\sum_i w_{ji} o_{pi} + \theta_j)}} \tag{2.28}$$

Since

$$\frac{\partial o_{pj}}{\partial net_{pj}} = o_{pj}(1 - o_{p,j}) \tag{2.29}$$

the corresponding error signal δ_{pj}, for an output unit is given by

$$\delta_{pj} = (t_{pj} - o_{pj}) o_{pj}(1 - o_{pj}) \tag{2.30}$$

and for an arbitrary hidden unit u_j is given by

$$\delta_{pj} = o_{pj}(1 - o_{pj}) \sum_k \delta_{pk} w_{kj} \tag{2.31}$$

2.1.5.6 Boltzmann Machine(BM)

The Boltzmann Machine(BM) [AHS87, HS86] is a parallel computational organization that is well suited to constraint satisfaction tasks involving large numbers of weak constrains. Since constrain-satisfaction problem involves strong constrains that must be satisfied, a variation is to use weak constrains that incur a cost when violated. Boltzmann Machine is designed for such a weak constrain problem, the energy of its global states is the cost we want to minimize.

Why call it 'Boltzmann Machine' In 'Canonical Ensemble', consider the system of N independent molecules with constant total energy E. While the amount of energy associated with any given particle cannot be determined, one can address the question on how the energy is distributed over the particles on average:

$$n_i = N * \frac{e^{-\beta * \epsilon_i}}{\sum_j e^{-\beta * \epsilon_j}} \tag{2.32}$$

n_i is the average number of particles that has energy ϵ_i, N is the total number of particles. β is constant. If the system is in contact with a heat bath at temperature T , it will eventually reach equilibrium and the probability that it is in the i_{th} micro-state, with energy ϵ_i, is given by the Boltzmann distribution:

$$p_i = \frac{e^{-\epsilon_i/k_B*T}}{Z} = e^{-\frac{\epsilon_i - A}{k_B*T}} \tag{2.33}$$

k_B is Boltzmann constant. T is the absolute temperature, thermodynamic β can be written as $\frac{1}{k_B*T}$, $Z = e^{\frac{-A}{k_B T}}$ is the normalizing factor and A is Helmholtz free energy.

In the Boltzmann Machine, every node has two states: on or off. This is the same as for a particle which has two energy states. Similarly, a network of units obeying this decision rule will eventually reach 'thermal equilibrium', thus, can also be simulated by Boltzmann distribution. This is why we call it 'Boltzmann Machine'.

Character of Boltzmann Machine Boltzmann Machine is composed of units that are connected to each other by bidirectional links. The states of units are on or off, which was used to represent the system accept or reject the unit hypothesis. The weight on a link represents a pairwise constrain of the two units. A positive weight means the two hypotheses tend to support each other, and vice versa. Link weights are symmetric, having the same strength in both directions [AHS87]. Many extensions are possible and have been studied, including cases where hidden and visible units to model continuous values [FH94, WRZH04].

If there are n units, each units can be on or off, then the system has totally 2^n global states. According to EBM, each global state can be assigned to a single number(energy) to represent it. The individual units are made to minimize the global energy [AHS87]. The energy is acted as the cost in the weak constrain problems. It can be interpreted as the extent to which the hypotheses violates the constrains.

The energy of a global configuration is defined as

$$E = -\sum_{i<j} w_{i,j} s_i s_j + \sum_i \theta_i s_i \tag{2.34}$$

where $w_{i,j}$ is the weights between units i and j. s_i represent the units on or off,($s_i = 1$ represent unit i is on, and vice versa). θ_i is a threshold.

A simple algorithm to find a local minimum energy is to switch each hypothesis whichever of its two states yields the lower global energy given the current states of the other hypotheses [AHS87, Hop82]. We use E_{off} to represent the global energy when the k_{th} unit is off(when $s_i = 1$), and E_{on} to represent the global energy when the k_{th} unit is on, then the energy gap between them is

$$\Delta E_k = E_{off} - E_{on} = \sum_{i=1, i\neq k}^{n} w_{k,i} s_i - \theta_k \tag{2.35}$$

Obviously, the strategy to determine the on or off state of a unit is to see whichever yields a lower global energy.

Threshold term θ_k can be omitted if we treat a link with weight $-\theta_k$ between unit k and an external unit (the $n+1$ unit) whose state is always on($s_{n+1} \equiv 1$).

One standard technology that used to minimize the energy is gradient descent: The values of the variables in the problems are modified in whatever direction reduces the energy. But gradient descent suffers from local minima problem that are not global optimal. [KGV83] uses 'simulated annealing'. This technology simulates the same procedure in physical analogy: To find low energy of a metal, the best strategy is first melt it and then to slowly reduce its temperature. Providing some noises can help getting out of local minima and allow jumps to higher energy [HS86].

Using the following formula, we can implement simulated annealing. If the energy gap between the 1 and the 0 states of the k unit is ΔE_k, then $s_k = 1$ with probability

$$p_k = sigmoid(\frac{\Delta E_k}{T}) \tag{2.36}$$

T acts like the temperature of annealing. Using 'simulated annealing' strategy, starting from a higher temperature and gradually reduce it, the system can go the fastest way to reach equilibrium.

Since the probability that the system finally falls in any global energy state follows a boltzmann distribution, so it can be easily derived that the relative probability of two global states also follows the Boltzmann distribution:

$$\frac{P_\alpha}{P_\beta} = e^{-\frac{E_\alpha - E_\beta}{T}} \tag{2.37}$$

If $T = 1$, we can get:

$$\log \frac{P_\alpha}{P_\beta} = -E_\alpha + E_\beta \tag{2.38}$$

Which indicates that the difference in their negative log probability is their energy gap.

Training Algorithm of BM According to equation 2.33, when the system goes to thermal equilibrium, the probability of the system in any global state is only determined by its energy. So the relationship between log probability of the system in any global

state and the weights is:

$$\begin{aligned}\frac{\partial \ln P_\alpha}{\partial w_{i,j}} &= \frac{\partial \ln \frac{e^{-\frac{E_\alpha}{kT}}}{\sum_\beta e^{-\frac{E_\alpha}{kT}}}}{\partial w_{i,j}} = \frac{\partial -\frac{E_\alpha}{kT} - \ln \sum_\beta e^{-\frac{E_\beta}{kT}}}{\partial w_{i,j}} \\ &= \frac{s_i^\alpha s_j^\alpha}{kT} - \frac{e^{-\frac{E_1}{kT}} * \frac{\partial -E_1/kT}{\partial w_{i,j}} + ... + e^{-\frac{E_n}{kT}} * \frac{\partial -E_n/kT}{\partial w_{i,j}}}{\sum_\beta e^{-\frac{E_\beta}{kT}}} \\ &= \frac{s_i^\alpha s_j^\alpha}{kT} - \sum_\beta P_\beta * \frac{s_i^\beta s_j^\beta}{kT}\end{aligned} \tag{2.39}$$

If we ignore the constant k, the equation becomes

$$\frac{\partial \ln P_\alpha}{\partial w_{i,j}} = \frac{1}{T} * (s_i^\alpha s_j^\beta - \sum_\beta P_\beta^- s_i^\beta s_j^\beta) \tag{2.40}$$

where s_i^α is the binary state of the i_{th} unit in the α_{th} global state and P_β^- is the probability, at thermal equilibrium, of global state α of the network when none of the visible units are clamped.

Visible units of BM are the interface between the environment and the network, it can be clamped into specific states, as the input of the environment to the network. The hidden units, are used to model the internal structure of the network.

The separate probability of each visible unit being active is the first order structure and can be captured by the thresholds of the visible units. The v^2 pairwise correlations between the v visible units constitute the second order structure and can be captured by the weights between pairs of units. All structure higher than second order can not be captured by pairwise weights between the visible units. An example is shown by [HS86]: Suppose we have three units, A B and C. The ensemble consists of four vectors: 1 1 0, 1 0 1, 0 1 1, 0 0 0. Each has the probability of 0.25. If we consider only first and second orders, this ensemble is indistinguishable from the ensemble in which all eight vectors occur with equal probability. So there must be some higher order constrains in the model.

The structure of an environment can be specified by giving the probability distribution over all 2^v states of the v visible units. Even if the whole network is totally connected the C_{v+h}^2 weights and $v + h$ biases, the model is insufficient to model the 2^v probabilities of the states of the visible units specified by the environment(unless the number of hidden units is exponentially large compared to the number of visible units). But we assume that if there are regularities in the environment, and if the network uses its hidden units to capture these regularities, it may achieve good match.

'KullBack-Leibler(K-L) Divergence' is used to measure the discrepancy between the network's internal model and the environment [HS86]:

$$G = \sum_{\alpha} P^{+}(V_\alpha) \ln \frac{P^{+}(V_\alpha)}{P^{-}V_\alpha} \tag{2.41}$$

Where $P^{+}(V_\alpha)$ is the probibility of the α_{th} state of the visible units which are determined by the environment(+ is used to indicate the states that are determined by the environment),and $P^{-}(V_\alpha)$ is the probability that the network is running freely. G is always larger or equal to 0. G is a asymmetric divergence. When trying to approximate a probability distribution, it is more important to get the probabilities correct for events that happen frequently than for rare events, so the match should be weighted by the actual probability[HS86].

The gradient of G with respect to weights is .

$$\frac{\partial G}{\partial w_{i,j}} = -\frac{1}{T}[p^{+}_{i,j} - p^{-}_{i,j}] \tag{2.42}$$

where $p^{+}_{i,j}$ can be explained as the probability, averaged over all the environmental inputs and measured at equilibrium that the i_{th} and j_{th} units are both on when the network is being driven by the environment, and $p^{-}_{i,j}$ the corresponding probability when the network is running freely. Then using gradient descent, weights are changed proportional to the difference between $p^{+}_{i,j}$ and $p^{-}_{i,j}$

$$\Delta w_{i,j} = \epsilon(p^{+}_{i,j} - p^{-}_{i,j}) \tag{2.43}$$

In minimizing G, the network is finding the set of weights that is most likely to have generated the set of environmental vectors.

Drawbacks of BM The nature of BM makes it very hard to train well. Collect equilibrium statistics grows exponentially with the machine's size, and with the magnitude of the connection strengths. Another problem is that connection strengths are more plastic when the units being connected have activation probabilities intermediate between zero and one, leading to a so-called 'variance trap'. The net effect is that noise causes the connection strengths to random walk until the activities saturate [BM]. If the connectivity is constrained, the learning can be made efficient enough to be useful for practical problems.

2.1.5.7 Restricted Boltzmann Machine(RBM)

Although learning is impractical in general Boltzmann machines, it can be made quite efficient in an architecture called the "Restricted Boltzmann Machine(RBM)" [Smo86, FH94], which does not allow intra-layer connections between hidden units. RBM is the most commonly used structure of deep learning. It is a bipartite graph with two layers: visible layer and hidden layer. Layers are connected with undirected weights. Each layer consists of stochastic binary units. Each state of RBM has an energy function, using stochastic binary states of visible units v and binary states of hidden units h, the energy of this state is defined as

$$E(v,h) = -\sum_{i,j} W_{i,j} v_i h_j - \sum_i v_i b_i - \sum_j h_j c_j \tag{2.44}$$

where b and c are biases in visible and hidden layers.

The probability of a joint configuration over both visible and hidden units depends on the energy of that joint configuration compared with the energy of all other joint configurations

$$p(v,h) = \frac{e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}} \tag{2.45}$$

Due to the constrain that there is no connection within the same layer, the posterior probability of hidden states given visible states are conditionally independent.

$$p(v_i = 1|h) = \sigma(\sum_j W_{i,j} h_j + b_i) \tag{2.46}$$

$$p(h_j = 1|v) = \sigma(\sum_i W_{i,j} v_i + c_j) \tag{2.47}$$

where σ is the sigmoid function($\sigma(x) = 1/(1+e^{-x})$), thus we can quickly get an unbiased sample from the posterior distribution when given a data vector. Commonly, it uses contrastive divergence(CD)[CPH05, Hin02] to learn RBM. CD aims to minimize the difference of two K-L divergences:

$$KL(p^0|p_\theta^\infty) - KL(p_\theta^n|p_\theta^\infty) \tag{2.48}$$

where P^0 is the distribution of data and P_θ^∞ is the equilibrium distribution defined by the model[Hin02].

2.1.5.8 Deep Belief Nets

RBMs can be stacked into multi-layers to get a deep network which can capture higher order correlations between the visible units [HOT06, HS06, TFW08]. The training algorithm of the deep network consists of pre-training and fine-tuning phases. More specifically, in pre-training phase, after training one RBM model, we can use their hidden activities as visible data for higher layer RBM model, thus can learn a second layer feature. It can be shown that adding an extra layer always improves a lower bound on the log probability that the model assigns to the training data, provided the number of feature detectors per layer does not decrease and their weights are initialized correctly [HOT06]. In fine-tuning phase, we make all the unit deterministic. Parameters from pre-training are retrained and adjusted in a small area though preforming gradient descent on them using back-propagation.

2.1.5.9 Deep Boltzmann Machine

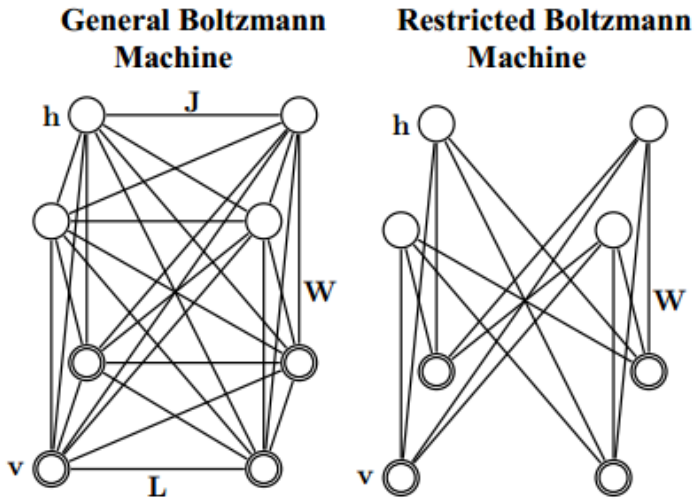


Figure 2.1: This is a picture of BM and DBM from [SH09a].**Left** is a general Boltzmann machine. The top layer represents a vector of stochastic binary hidden features and the bottom layer represents a vector of stochastic binary visible variables. **Right** is a restricted Boltzmann machine with no hidden-to-hidden and no visible-to-visible connections.

[SH09a, SH10] introduced a Deep Boltzmann Machine(DBM). Unlike Deep Belief Net, DBM have undirected weights between adjacent layers. It is interesting for several reasons [SH10]. First, like DBNs, DBM has the ability to learn internal representations that capture every complex statical structure in the higher layers. This is a promising way of solving object and speech recognition problems [Ben09a, rMDH, rMDH12]. Second, like DBNs, if DBMs are learned in the right way, there is a very fast way to initialize

the states of the units in all layers by simple doing a single bottom up pass using twice the weights to compensate for the initial lack of top-down feedback. Third, unlike DBNs and unlike many other models with deep architectures [HlBL07, VLBM08, SOP], the approximate inference procedure, after the initial bottom up pass, can incorporate top down feedback, allowing DBM to use higher level knowledge to resolve uncertainty about the intermediate level features.

2.1.5.10 Auto-Encoder

The aim of an auto-encoder[BLP+07, BK, VLBM08] is to learn a compressed representation (encoding) for a set of data. This means it can be used for dimensionality reduction. Basically, an auto-encoder consists of one input layer, one output layer and one or some hidden layers. From the input layer to the middle hidden layer is the encoding procedure, from the middle layer to the output layer is the decoding procedure. Its objective function is to minimize the reconstruction error of the input data. Generally, there are two types of objective functions that are used, either minimize the square error between input and output or minimize the cross entropy between them.

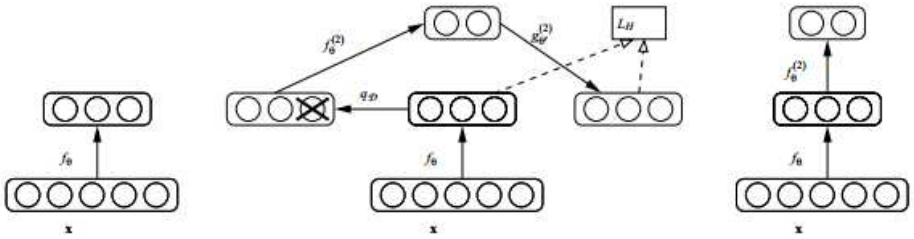


Figure 2.2: This is a picture of DA and SDA from [VLBM08]. After training a first level denoising autoencoder(left), its learnt encoding function f_θ is used on clean input (left). The resulting representation is used to train a second level denosing autoencoder (middle) to learn a second level encoding function $f_\theta^{(2)}$. From there, the procedure can be repeated (right).

Denosing Auto-encoder(DA) The idea behind DA is simple, in order to force the hidden layers to find some useful features instead of being the same of the inputs, we input to the DA an corrupted version and force it to recover the original clean data. A simple denosing auto-encoder training criterion is equivalent to matching the score (with respect to the data) of a specific energy based model to that of a non-parametric Parzen density estimator of the data [Vin11].

Stacked Denosing Autoencoder(SDA) The Stacked Denoising Auto-encoder (SDA) is an extension of the stacked auto-encoder [Bengio07] and it was introduced in [Vincent08]. The basic training algorithm is like training a DBN, first greedy layer-wise pre-training, followed by fine-tuning the network. But there is something different, each time we stacked one layer DA, we use the representation of un-corrupted version of the data as input to a higher layer instead of using the representation of corrupted ones.

2.1.6 Applications of Deep Learning

At 2006, Hinton's group[HOT06] lighted the first fire on the grassland of deep Learning, the whole world went into an revolution. Deep learning got its development as quick as thunder. Researchers applied Deep Learning to various fields, including sparse feature learning [Hin07, RBL07, LEN08, PCL06, AWG10, LBRN07, SKC+11, HlBL07, HlBL07, LGRN09, GCB12, VLBM08, SAL12, HSK+12, MH10], Nature Language Processing (NLP) [MH08, CW08, SMH11, MT12], topic model [DAL12, SH], transfer learning [MD10, DB11, CXWS12, MDG+12], metric learning [BSJ11], kernel learning [SH07, YXG]. speech recognition [rMDH12, HDY+12], data compression [HS06]. audio detection and generation [BLBPVtM, MPLB11, BMEM10]. At the same time, some authors showed new ideas of how to improve deep models and gave variety of new training algorithms [PH01, HS83, NCKN11, RSMH11].

2.1.6.1 Sparse Model and Feature Learning

The first sparse RBM is proposed by [LEN08]. The reason that bring sparseness into RBM is that RBM tends to learn distributed, non-sparse representations, based on the results from sparse coding [HH02, LBRN07], ICA [BS97] and energy based models [PCL06]. Sparseness plays a key role in learning gabor-like filters. The basic idea of training a sparse RBM is to force the hidden unit activations to be sparse. Basically, there are two ways to do this. The first way is to add some regularization term. [LEN08] adds a regularization term that penalizes a deviation of the expected activation of the hidden units from a fixed level that is provided by ourself, it uses a square error loss. Hinton provides another cross entropy loss on spareness $\rho \log q_i + (1-\rho)\log(1-q_i)$, where q_i is the mean activation in hidden layer and ρ is the sparseness we specify [Hin10]. The second way is to change the formulation of RBM, like Cardinality-RBM [STS+12], Boltzmann Perceptron[Kap95] . [STS+12] provides another formulation of sparse RBM. Instead of using the regulation term in the object function, it forms the sparseness into the formulation of RBM. The formulation of CRBM is defined as

$$P(\mathbf{v},\mathbf{h}) = \frac{1}{Z} exp(\mathbf{v}^T W \mathbf{h} + \mathbf{v}_v^{\mathbf{b}} + \mathbf{h}^T \mathbf{b}_h) * \psi_k(\sum_{j=1}^{H_h} h_j) \tag{2.49}$$

where ψ_k is a potential given by $\psi_k(c) = 1$ if $c \leq k$ and 0 otherwise. This conditional distribution $P(h|v)$ assigns a non-zero probability mass to a vector h only if $h \leq k$. [Kap95] is a similar model to CRBM, which also introduces a term in the energy function to promote competition between units. Another similar work is RBF [LBT10] which uses k groups of multinomial hidden units. Honglak Lee's group provides an efficient sparse coding algorithm [LBRN07] which can also be well suited for deep learning tasks.

When the basic RBM is got, deep architecture can also be built to extract useful representation. Hinton stacked the RBMs to form a DBN that can be used as a generative model as well as a discriminative model [HOT06, Hin07] which can be used to learn multiple layers of representation. [RBL07] introduces a Sparse Encoding Symmetric Machine(SESM) that has a set of linear filters in both encoder and decoder to learn sparse features in DBN.

Besides DBN, other deep models can also be used to extract features and learn useful representation. In order to deal with full-sized, high dimensional images, [LGRN09] use Convolution DBN which is translation-invariant. It uses probabilistic max-pooling to shrinks the representations of higher layers and allows higher layer units to cover larger areas of input, it benefits from the fact that certain convolutional pooling architectures can be inherently frequency select and translation invariant[SKC+11].

[VLBM08] uses de-nosing auto-encoders to extract and compose robust features. [HlBL07] is also based on encoder-decoder paradigm to build sparse, shift-invariant features. [HlBL07] combines sparse coding and feature learning, it uses topographically-organized feature map to extract locally invariant image descriptors for image recognition. [PCL06] is another approach to learn sparse, over-complete features using encoder-decoder paradigm.

Other Variants and Improvements CD [Hin02, CPH05] is the first efficient and quick learning algorithm to train RBM. When estimating the negative gradient of the log likelihood, it is designed for at least the direction of the gradient estimate is accurate, even when the size is not. Later, [Tie08, TH09] introduces Persistent CD(PCD) that uses a persistent Markov chain to estimate the model's estimation. In order to improve the stochastic gradient descent(SGD) that used to train deep models, [LNC+11] introduces Limit memory BFGS(L-BFGS) and conjugate gradient(CG) that can simplify and speed up the pre-training step of deep learning.

Another direction that speeds up deep learning and makes it scalable is to use parallel technology like GPU or distributed computing [RMN09], those kinds of technology make large scale deep learning practical[LRM+12].

2.1.6.2 Improvements of Deep Models

Besides the Convolution RBM, Cardinality RBM, DBN, DBM, SDA, there are also some other models. Helmholtz Machine [DHNZ95] is an old model used to describe a class of neural networks which learn the hidden structure to create a generative model. [TSH12] provides a deep lambertian network that combine DBN with Lambertian reflectances. [RSMH11] combined DBN with gated Markov Random Field(MRF). Gaussian RBM can be used to model real value pixels and was introduces by [Hin10, Kri09]. Instead of using Gaussian RBM in the first layer of DBN to model real values, [RSMH11] uses a gated MRF called mPot [RMH10] to model continues pixel values since it is capable of generating more realistic high-resolution images. [NH09] uses a third-order Boltzmann machine [Sej86] for top-level DBN and uses it for 3D object recognition. [SH] uses replicated softmax layer in the bottle to model word count vectors. [STT11] provides Hierarchical-Deep models that integrates deep learning models with structured hierarchical Bayesian models with can learn novel concepts from very few training examples. [WRZH04] introduces exponential family harmoniums and use them for IR. [HSK$^+$12] introduces the trick of 'dropout' to prevent complex co-adaptation on training data and give big improvements when the neural network is trained on small training set. Temporal RBM [SHT08] is an extension of RBM in temporal domain and is used to model high dimensional sequences. [SH12] introduces a better way to train DBM. [LEV09] extends the denosing auto-encoder by adding asymmetric lateral connections between hidden units and shows significantly improvement on the classification performance. To model the partition function of RBM is always a hard problem, [SM08] shows that AIS [Nea98] can be used to efficiently estimate the partition function of an RBM, later [SH] extends AIS to estimate the partition function of Replicated Softmax Model.

Another survey that explores some training algorithms and explores different RBM input distributions can be found at [LBLL09]

2.1.6.3 Training Algorithms

In this section, I would like to introduce some classic training algorithms that are used in deep learning.

Wake-Sleep Algorithm [HDFN95] introduces a wake-sleep algorithm. In the wake sleep algorithm, the goal is to learn representations that are economical to describe but allow the input to be reconstructed accurately. There are two phases, 'wake' phase and 'sleep' phase. In wake phase, neurons are driven by bottom up recognition connections and generative connections are adapted to increase the probability that they could reconstruct the correct activity vector in the layer below. In the sleep phase, recognition weights are adapted to increase the probability that they produce the correct activity vector in the layer above. Wake-Sleep algorithm uses the local delta rule to changes weights.

Contrastive Divergence(CD) CD was first proposed by Hinton [Hin02] as a fast training algorithm minimize the difference of two K-L divergences [CPH05, Hin02, BD09]. There are a lot advantages of combine n individual expert models to form a Product of Experts. Combine n individual expert model as follows:

$$p(d|\theta_1, \theta_2, ...\theta_n) = \frac{\Pi_m f_m(d|\theta_m)}{\sum_c \Pi_m f_m(c|\theta_m)} \tag{2.50}$$

$f_m(d|\theta_m)$ is the probability of d under model m. θ_m is all the parameters of model m. So if we want to maximize the log likelihood of each observed data vector d, we must calculate the derivative of it:

$$\frac{\partial \log P(d|\theta_1, \theta_2, ...\theta_n)}{\partial \theta_m} = \frac{\partial \log f_m(d|\theta_m)}{\partial \theta_m} - \sum_c p(c|\theta_1, \theta_2, ...\theta_n) \frac{\partial \log f_m(c|\theta_m)}{\partial \theta_m} \tag{2.51}$$

Be informed that this equation is the same of equation 2.39.

According to Hinton [Hin02], maximize the log likelihood, can use Gibbs sampling that can alternate between parallel updates of the hidden and visible variable. To get an unbiased estimate of the gradient, it is necessary for the Markov chain to converge to the equilibrium distribution. Here, problems come, first of all, it is computationally infeasible to approach the equilibrium distribution before taking samples, second, samples from the equilibrium distribution generally have high variance[Hin02].

Maximizing the log likelihood of the data(averaged over the data distribution) is equivalent to minimizing the K-L divergence between the data distribution P^0 and the equilibrium distribution over the visible variable P^∞ that is produced by prolonged Gibbs sampling.

$$\begin{aligned} KL(P^0||P_\theta^\infty) &= \sum_d P^0(d) \log \frac{P^0(d)}{P_\theta^\infty(d)} = \sum_d P^0(d) \log P^0(d) - \sum_d P^0(d) \log P_\theta^\infty(d) \\ &= -H(P^0) - < \log P_\theta^\infty >_{P^0} \end{aligned} \tag{2.52}$$

Equation 2.51, averaged over the data distribution, is :

$$\begin{aligned} < \frac{\partial \log P(d|\theta_1, \theta_2, ...\theta_n)}{\partial \theta_m} >_{P^0} &= < \frac{\partial \log f_m(d|\theta_m)}{\partial \theta_m} >_{P^0} - < \sum_c p(c|\theta_1, \theta_2, ...\theta_n) \frac{\partial \log f_m(c|\theta_m)}{\partial \theta_m} >_{P^0} \\ &= < \frac{\partial \log f_m(d|\theta_m)}{\partial \theta_m} >_{P^0} - < \frac{\partial \log f_m(d|\theta_m)}{\partial \theta_m} >_{P^\infty} \end{aligned} \tag{2.53}$$

There is a simple and effective alternative to maximum likelihood learning that eliminate almost all of the computation required to get samples from the equilibrium distribution and also eliminates much of the variance. According to Hinton [Hin02], instead

of minimizing $P^0||P^\infty$, we minimize the difference between $P^0||P^\infty$ and $P^1||P^\infty$, where P^1 is the distribution over one step reconstruction of the data vectors generated by one full step Gibbs sampling.

Through doing this, we reduce the tendency of the chain to wander away from the initial distribution on the first step. Since Markov chains have nonzero probability in all transitions, $P^0 = P^1_\theta$ implies that $P^0 = P^\infty_\theta$. If the distribution does not change on the first step, it must already be at equilibrium.

For mathematical explanation:

$$\begin{aligned} -\frac{\partial(P^0||P^\infty_\theta - P^1||P^\infty_\theta)}{\partial\theta_m} &= -\frac{\partial(-H(P^0) - <\log P^\infty_\theta>_{P^0} + H(P^1_\theta) + <\log P^\infty_\theta>_{P^1_\theta})}{\partial\theta_m} \\ &= \frac{\partial(<\log P^\infty_\theta>_{P^0} - <\log P^\infty_\theta>_{P^1_\theta} - H(P^1_\theta))}{\partial\theta_m} \end{aligned} \tag{2.54}$$

Then according to equation 2.53, we get:

$$-\frac{\partial(P^0||P^\infty_\theta - P^1||P^\infty_\theta)}{\partial\theta_m} = <\frac{\partial\log f_m(d|\theta_m)}{\partial\theta_m}>_{P^0} - <\frac{\partial\log f_m(d|\theta_m)}{\partial\theta_m}>_{P^1_\theta} - \frac{\partial H(P^1_\theta)}{\partial\theta_m} \tag{2.55}$$

And

$$-\frac{\partial H(P^1_\theta)}{\partial\theta_m} = \frac{\partial P^1_\theta}{\partial\theta_m}\frac{-H(P^1_\theta)}{\partial P^1_\theta} = \frac{\partial P^1_\theta}{\partial\theta_m}\frac{-H(P^1_\theta) - <\log P^\infty_\theta>_{P^1_\theta}}{\partial P^1_\theta} = \frac{\partial P^1_\theta}{\partial\theta_m}\frac{\partial(P^1_\theta||P^\infty_\theta)}{\partial P^1_\theta} \tag{2.56}$$

So we get:

$$-\frac{\partial(P^0||P^\infty_\theta - P^1||P^\infty_\theta)}{\partial\theta_m} = <\frac{\partial\log f_m(d|\theta_m)}{\partial\theta_m}>_{P^0} - <\frac{\partial\log f_m(d|\theta_m)}{\partial\theta_m}>_{P^1_\theta} + \frac{\partial P^1_\theta}{\partial\theta_m}\frac{\partial(P^1_\theta||P^\infty_\theta)}{\partial P^1_\theta} \tag{2.57}$$

As shown in [Hin02]. Hinton also proves that the last term of equation 2.57 can be omitted because it is small and seldom opposes the resultant of the other two terms.

2.2 Learning to Hash Survey

In this section, I will introduces the latest hashing algorithms in the field of learning to hash.

2.2.1 Nearest Neighbor Search

Nearest Neighbor Search(NNS) is an optimization problem for finding closest points in metric spaces. It is defined as this: given a collection of n points, build a structure which, given any query point, reports the data point that is closest to the query [AI08].

Nearest neighbour search problem is an import problem in a variety of applications, including data mining [FPSSU96], pattern recognition and classification [CH06], machine learning [CS93], data compression [GG91], multimedia search [FSN+95], document retrieval [SH09b].

Methods for fast nearest neighbor retrieval are generally broken down into two branches. One group partitions the data space recursively, like R-tree [MNPT05], K-D tree [Ben75], M-tree [CPZ97]. When the dimensionality is high, 'Curse of Dimensionality' will degrade these algorithms to linear time algorithms in the worst case. Another group is hashing based methods, which maps the data into a low dimensional Hamming space and do fast retrieval in that space [GKVL12, GIM99, Lee12, ZWS11, XWL+11].

Later, in order to overcome the running time issues, some approximation methods are proposed [AMN+98, IM98, HAYSZ11]. The basic idea of approximation is that the algorithm is allowed to return a point whose distance is within some extend to the real answer, because sometimes, an approximate point is almost as good as the exact one.

2.2.2 Hashing Methods

In this section, I mainly introduce some existing important hashing methods that are widely used.

2.2.2.1 Locality Sensitive Hashing(LSH)

Locality Sensitive Hashing(LSH) [IM98] is an algorithm for solving nearest neighbor search problem in high dimensional space. It can solve approximate NNS and exact NNS problem and has theoretical guarantee for sub-linear search time. The basic idea behind LSH is that, for each search, the probability of collision is much higher for objects that are close to each other than for those that are far apart [AI08]. The LSH algorithm has been applied to a variety of areas, such as web clustering [HGI00], computational biology [**?**]. LSH has also been extended to various metric measures, such as l_1 norm [LLR95], l_p norm [DIIM04], kernelized version [KG09], learned metrics [JKG08],

arccos similarity [Cha02], Jaccard similarity [Bro97], image kernels [Gra07]. See [IM98] for more theoretical analysis of LSH.

But LSH will need a large code length to get good performance, which is very inefficient.

2.2.2.2 RBM and Semantic Hashing(SemanH)

Rather than using random projection to define bits, [TFW08] uses an auto-encoder to form the hash codes. It stacked some RBMs to form a deep structure and use NCA for back propagation in order to preserve the neighbour structure in the data. It solves the problem of both storing millions of images into memory and quickly finding similar images to a target image. In [TFW08], RBMs are shown to have better performances than both LSH and Boosting SSC.

The basic idea of Semantic Hashing[SH09b] is that each item is represented as compact binary codes so that similar items will have similar binary codewords in memory. Retrieval similar neighbours is done by find all items that are within a small Hamming distance. It produces a short-list of similar documents in a time that is independent of the size of the document collection and linear in the size of the short-list. It uses a conditional 'constrained' Possion distribution to model documents and Bernoulli distribution to model the hidden topics. The result of Semantic Hashing on document retrieval is much better than some tradition methods like LSH, Latent Semantic Analysis(LSA) [DDF$^+$90].

Both [TFW08] and [SH09b] use deep learning techniques to do hashing . They are new approaches in the application of deep learning. But one big problem is that the training time is intolerable, this is the common problem of deep learning methods. To train a model well, one need to have good search in the parameter spaces and tune millions of parameters. It needs a large training time and training samples.

2.2.2.3 Spectral Hashing(SH)

In [WTF08], the author first proposed the question of 'What makes a good code for hashing'. The code should be: First, easily computed for 'out of sample' extension. Second, requires a small number of bits to code full dataset. Third, Maps similar items to similar codewords.

The formulation of SH is as follows,

$$minimize : \sum_{i,j} W_{i,j} ||y_i - y_j||^2 \tag{2.58}$$

$$subject\ to : y_i \in \{-1, +1\}^k \tag{2.59}$$

$$\sum_i y_i = 0 \tag{2.60}$$

$$\frac{1}{n} \sum_i y_y y_i^T = I \tag{2.61}$$

Where $W_{i,j}$ is the affinity matrix between original points. y_i and y_j are code words. The constraint $\sum_i y_i = 0$ requires each bit to fire 50% of the time, and the constraint $\frac{1}{n}\sum_i y_y y_i^T = I$ requires the bits to be uncorrelated. After spectral relaxation, the problem can be solved by solving an eigenvector-eigenvalue problem. For out of sample extension of new coming data, the author assumes some distribution of data, e,g uniform distribution, which is one of the biggest drawbacks of this method.

Results [WFT12] show that SH does a very bad job of approximating the distance to far away points and collapses all of them to a similar distance. Another big drawback of SH is that it often performs well when the length of codewords is short. But later, the same authors of SH gave a complementary of SH, known as MDSH.

2.2.2.4 Multidimensional Spectral Hashing (MDSH)

In [WFT12], the authors proposed a new learning to hash framework. They found that performances of existing learning to hash methods can change drastically depending on the definition of ground-truth neighbors. So they propose a new formulation for learning binary codes which seeks to reconstruct the affinity between datapoints rather than their distances. The solution is similar to SH at some extend, but different. First, it tries to approximate original affinity using code words, and also has theoretical guarantee of convergence with the increasing length of code works, which is a complementary of SH, since SH general performs bad with the code words increase [RL09]. Second, the independence of bits are implicitly required. Third, mapping from a datapoint to its code is not restricted to any particular functional form, instead, best codes are got by binary matrix factorization of the affinity matrix.

2.2.2.5 Shift Invariant Kernel Hashing(SIKH)

SIKH [RL09] is another distribution-free encoding scheme based on random projections that the expected Hamming distance between the binary codes of two vectors is related to a shift-invariant kernel between the vectors. Although based on random projection, the embedding is highly non-linear.

2.2.2.6 Density Sensitive Hashing(DSH)

DSH [LCL12] is another extension of LSH. Instead of using random projection, DSH guides the projection. First, it uses k-means to roughly partition the data set into k groups, then for each pair of adjacent groups, DSH generates one projection vector which can well split the two corresponding groups. Finally, DSH selects the final projections based on an entropy based principle.

2.2.2.7 Graph Hashing(GH)

Graph hashing [LWKC11] is based on Spectral Hashing. Instead of assuming a separate data distribution like SH, GH solves it using anchor graphs. The basic idea of GH is to use the m anchors to approximate the true adjacency matrix in the formulation of SH. The original SH problem is converted to solving the graph Laplacian eigenvectors. For out of sample extension, the author uses Nystrom method [WS01, BDLR$^+$04]. In hashing procedure, the author uses a hierarchical way.

2.2.2.8 Minimal Loss Hashing(MLH)

In order to learn w, MLH [NF11] tries to minimize the empirical loss over training pairs.

$$L(w) = \sum_{i,j \in S} L(b(x_i, w), b(x_j, w), s_{i,j}) \tag{2.62}$$

As to the loss function, the author simple chooses a hinge loss function. MLH is solved using Structural SVM [YJ09, JFY09], since it benefits from the convex and concave bound.

2.2.2.9 Iterative Quantization Hashing(ITQ)

The basic formulation of ITQ [GL11] is the same with [WTF08, WKC10a]. The difference is that it adds another binary quantization procedure. It tries to minimize the quantization error of mapping data to the vertices of the binary hypercubes through orthogonally transformation.

ITQ can be implemented as unsupervised method if we use PCA as embedding method. It tries to learn an orthogonal rotation matrix to refine the initial projection matrix learned by PCA so that the quantization error of mapping the data to the vertices of binary hypercubes is minimized.

ITQ can also be implemented as supervised method if we provide label information and use CCA [HSSSt07] as embedding method.

2.2.2.10 Binary Re-constructive Embedding(BRE)

BRE [KD] uses a loss function that penalizes the difference between Euclidean distance in the input space and the Hamming distance between binary codes. Instead of using standard random projection like LSH, BRE uses a kernel form to generate a sequence of hash functions that are dependent on one another.

2.2.2.11 Isotropic Hashing(IsoH)

Isotropic Hashing [KL12] is another extension of ITQ. PCA based hashing [GL11] adopts principle component analysis (PCA) to learn the projection functions. ITQ [GL11] tries to learn an orthogonal rotation matrix to refine the initial projection matrix learned by PCA so that the quantization error of mapping the data to the vertices of binary hypercubes is minimized. Isotropic Hashing try to learn a projection function that can produce projected dimensions with equal variance. There are another big difference compared with ITQ. ITQ quantizes the PCA results, then learns the orthogonal matrix based on the first step. Basically, it contains two separate steps, thus its goal can not be formulated by a single objective function. While, Isotropic Hashing, directly learns the orthogonal matrix from the eigenvalues and eigenvectors of PCA, it has a single objective function to minimize, which makes the whole learning procedure more smooth.

2.2.2.12 Kernelized LSH(KLSH)

KLSH [KG09] is the extension of LSH with kernel functions. It makes LSH and its variants search data with a number of powerful kernels, including many kernels designed specifically for image comparisons [Var07, ZBMM06, ZMLS07],as well as some well used functions like Gaussian RBF kernel.

2.2.2.13 Self-Taught Hashing(STH)

STH [ZWCL10] is another extension of SH with two tiny different tips. First, convert global affinity matrix in SH to a local similarity matrix. Second, in order to solve the 'out of sample extension', STH uses codes got from the training procedure as labels and treat the coding procedure as binary classification. STH is used at [ZWCL10] for text retrieval.

2.2.2.14 Semi-Supervised Hashing(SSH)

The formulation of SSH [WKC12] is easy to understand. Given some neighbor point pairs M and non-neighbor point pairs C, the objective function is to maximize the difference between correctly classified pairs and wrongly classified pairs by each bit:

$$J(H) = \sum_{k}\{\sum_{(x_i,x_j)\in M} h_k(x_k)h_k(x_j) - \sum_{(x_i,x_j)\in C} h_k(x_k)h_k(x_j)\} \tag{2.63}$$

For $h_k(x_i)$, they just use the simplest form $h_k(x_i) = sgn(w_k^T x_i + b_k)$.
After some optimization, the problem is converted to an eigenvalue problem.

2.2.2.15 Semi-Supervised Sequential Projection Learning to Hash(S3PLH)

S3PLH[WKC10b] is an improvement version of SSH. Following the formulation of SSH, the problem is converted to an eigenvalue problem. But the solution of SSH is sensitive to the choice of the penalty coefficient. [WKC10b] proposes a sequential projection learning method. The hash function is learned iteratively such that at each iteration, pairwise label matrix is updated such that more weights is given to the point pairs that violates the previous hash function.

This method can also be implemented in an unsupervised way, named USPLH [WKC10b]. To get the similarity pairs and non similarity pairs, the author creates pseudo pairwise labels in order to prevent that short distance point pairs between threshold are given different bits and long distance point pairs in one side of threshold are given the same bits.

2.2.2.16 Kernel-Based Supervised Hashing(KSH)

KSH [LWJ+12] is another extension of SSH, it uses a kernel formulation to map data in compact binary codes whose Hamming distances are minimized on similar pairs and simultaneously maximized on dissimilar pairs. The difference is that it uses a kernel form and plays a trick on manipulate the binary coeds.

Previous supervised methods like MLH [NF11] and BRE [KD] try to control the Hamming distances in the code space such that they can meet the supervised information. But it is difficult to optimize Hamming distance that are non-convex and non-smooth. While, KSH manipulates the code inner products, implicitly optimize in Hamming distance.

2.3 Search In Hamming Space

Basically, after we map the data into Hamming spaces, there are two ways to retrieve at Hamming space. (i)Hamming ranking - all the points in the database are ranked according to their Hamming distance from the query and the desired neighbors are returned from the top of the ranked list. (ii) Hash lookup - a lookup table is constructed using the database codes, and all the points in the buckets that fall within a small Hamming radius of the query are returned. The complexity of Hamming ranking is linear and constant time for hash lookup tables.

If hash lookup tables is well built, it will yield a dramatic increase in search speed compared to an exhaustive linear scan in Hamming space. To find near neighbors, one need to check all hash buckets within some hamming ball round the query. Here comes the question, the number of hash buckets grows near-exponentially [NPF12] with the search radius. Even with a small search radius, the number of buckets to examine may be larger than the number of items in the database, hence slower than linear scan.

2.3.0.17 Multi Index Hashing (MIH)

MIH [NPF12] is an approach for fast exact K-nearest neighbor search in Hamming space. The key idea of MIH is quite simple. Consider a binary string **a** and **b**, both has l bits. Partition them into m disjoint sub-strings, if **a** and **b** differ by t bits, then at least one of their m sub-strings must differ by at most $\lfloor t/m \rfloor$ bits. That means instead of comparing at l bits, we can search at $\lfloor l/m \rfloor$ bits at a Hamming radius $\lfloor t/m \rfloor$. The method is clever and ingenious.

2.3.0.18 Manhattan Hashing(MH)

Current hashing work is typically divided into two parts. First, projection, second, quantization. Most of the work uses single bit quantization. GH uses a hierarchical way to form multi bits quantization. Manhattan Hashing [KLG12] is another approach. It encodes each projected dimensional with multiple bits of natural binary code(NBC) and then Manhattan distance between points are calculated. [KLG12] shows some advantages of using Manhattan distances for such kinds of work. While, how to do quantization and how to do retrieval are still open problems and have a long way to go.

2.3.0.19 Other methods

There are some other hashing methods, such as Parameter Sensitive Hashing [SVD03], Coherency Sensitive Hashing[KA11], Entropy-Based Hashing(SPEC) [LRY10], for those interested, please refer to the references.

2.4 Conclusion

I attached conclusion tables (figure 2.3 to figure 2.5) about hashing algorithms.

Methods	Super-vised	Semi-Supervised	Un-Super-vised	Linear Projection	Non-linear Projection	Entropy based	Kernel method
LSH			√	√			
SH			√	√			
SIKH			√	√			√
BRE			√		√		√
USPLH			√	√		√	
BoostSSC	√						
RBMs	√				√		
SPEC	√				√	√	
S3PLH	√			√		√	
MLH	√			√			
SSH		√		√		√	
ITQ	√		√	√			
KLSH			√		√		√
DSH			√	√		√	
RR			√	√			
IsoHash			√	√			
MDSH			√		√		
GH			√	√			
STH			√	√			
SemanH	√				√		
KSH	√				√		√

Figure 2.3: classification based on supervised method and projection method. USPLH from [WKC10b], SPEC from [LRY10], RR from [GL11]

Methods	Random projection	Data dependent projection
LSH	√	
SH		√
SIKH	√	
BRE		√
USPLH		√
PSH		√
RBMs	√	
SPEC		√
S3PLH		√
MLH		√
SSH		√
ITQ		√
KLSH	√	
DSH		√
RR		√
IsoHash		√
MDSH		√
GH		√
STH		√
SemanH	√	
KSH		√

Figure 2.4: Data independent projection or data dependent projection

Methods	Hash function	Object function
LSH	$sgn(w_k^T x + b_k)$ $w\ from\ p\ stable$	
SH	$sgn(\sin(\frac{\pi}{2} + \frac{k\pi}{b-a}x))$	Minimize $\sum_{i,j} W_{i,j}\|\|y_i - y_j\|\|^2$ With constrains
SIKH	$sgn(\cos(kw * x))$ $w\ is\ random\ chosen$	
BRE	$h_p(x) = sign(\sum_{q=1}^{s} W_{p,q} k(x_{p,q}, x))$	$\min(l_{bre}(m_{i,j} - d_{i,j}) = (\frac{1}{q}m_{i,j} - \frac{1}{2}d_{i,j})^2)$
USPLH	$sgn(w_k^T x + b_k)$ $b_k = mean\ thresholding$	max $\frac{1}{2}tr\{W^T MW\}$ $M = X_l S X_l^T + \mu X X^T$
BoostSSC	$h_n(x_i) = e_n^T x_i > T_n$	$\min \sum_{k=1}^{K} w_n^k (z_k - f_n(x_i^k, x_j^k))^2$ $f_n(x_i^k, x_j^k) = \alpha_n[(e_n^T x_i > T_n) = (e_n^T x_j > T_n)] + \beta_n$
RBMs	Neural Network	NCA
SPEC		$\min L_T = J_T - J_{T-1}$ $J_T\ is\ cross\ entropy$
S3PLH	$sgn(w_k^T x + b_k)$ $b_k = mean\ thresholding$	max $\frac{1}{2}tr\{W^T MW\}$ $M = X_l S X_l^T + \mu X X^T$
MLH	$b(x, w) = thr(Wx)$	Minimize $L(w) = \sum_{(i,j)\in S} l(b(x_i, w), b(x_j, w), s_{i,j})$ l is hinge loss
SSH	$sgn(w_k^T x + b_k)$ $b_k = mean\ thresholding$	max $\frac{1}{2}tr\{W^T MW\} - \frac{\gamma}{2}\|W^T W - I\|^2$ $M = X_l S X_l^T + \mu X X^T$
ITQ	$sgn(w_k^T x)$ $for\ zero\ centered\ data$	Min quantization error $Q(B, R) = \|\|B - VR\|\|^2$
KLSH	$h(\emptyset(x)) = sign(\sum_{i}^{s} w(i)k(x, x_i))$	
DSH	$sgn(w_k^T x + b_k)$	Maximize the entropy provided by each bit
RR	$sgn(w_k^T x)$ $for\ zero\ centered\ data$	
IsoHash	$sgn(w_k^T x + b_k)$	Make the variances for different PCA dimensions are isotropic
MDSH		max $(Y, \Lambda) = \arg min\|\|W - Y\Lambda Y^T\|\|_F^2$ with constrains
GH	$sgn(w_k^T x + b_k)$	Minimize $\frac{1}{2}\sum_{i,j} A_{i,j}\|\|y_i - y_j\|\|^2$ With constrains
STH	$sgn(w_k^T x + b_k)$	Minimize $\frac{1}{4}\sum_{i,j} W_{i,j}\|\|y_i - y_j\|\|^2$ With constrains
SemanH	Neural Network	Auto-encoder
KSH	$h(\emptyset(x)) = sign(\sum_{i}^{s} w(i)k(x, x_i))$	Minimize $Q(A) =$ $\|\|\frac{1}{r}sgn(KA)(sgn(KA))^T - S\|\|_F^2$

Figure 2.5: Project function and objective function

Chapter 3

Comparison of Hashing Algorithms

In this chapter, I will show comparison experiment on various hashing algorithms. I focus on Euclidean search on various datasets. 13 widely used hashing methods have been chosen for the experiments. For Euclidean search, we define ($\ell_2@n\%$): top $n\%$ nearest neighbors of the query instance retrieved using the exact Euclidean distance on the target view will be treated as the ground truth relevant images in the retrieval task [WTF08, TFW08].

3.1 Parameters

In all the training data, I randomly split 20% of the training data as validation data. All parameters are choose by cross validation. For MLH, we use the original parameter choosing method with hinge loss. For ITQ, number of iteration is chosen from $[10, 50, 100]$ and use PCA for embedding. For USPLH, η is chosen from $[0.125 : 0.125 : 1]$, number of pseudo labels is from $[ntraining/8 : 200 : ntraining/4]$. For BRE, hash size is chosen from $[50, 100]$, number of data to form kernel is chosen from $[200, 400]$. For isoHash, PCA dimension is 150(using 100 for notredame dataset since its dimension is 128). LP method is chosen in isoHash. number of iteration is chosen from $[10, 30, 50]$. For MDSH, σ is chosen from $[0.1, 1, 10]$, ϵ is chosen from $[0, 1, 0.5]$, outlier is from $[10 : 3 : 20]$. For GH, we use two layer structure. Number of anchor is 200, number of nearest anchors is chosen from $[2, 5]$. σ=0 as suggested by author. For SIKH, γ is chosen from $[0.01, 0.1, 1, 10, 100, 150, 200]$. For KLSH, number of samples for kernel is chosen from $[100, 200, 300]$, number of Gaussian approximation is chosen from $[10, 50, 100]$, we use RBF kernel. σ is chosen from $[1, 5, 10, 50, 100]$. For SH, σ is chosen from $[0.1, 1, 10]$.

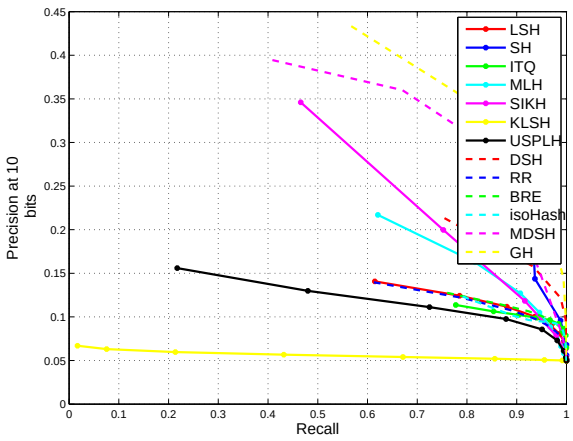
We use cross validation to choose parameters that can generate the best MAP (which I define in 'Evaluation Criteria' part) value. All the experiment I report here are executed three times and results are mean value.

3.2 Evaluation Criteria

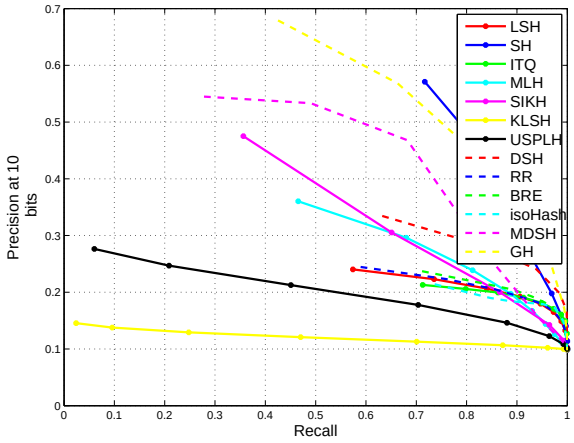
We show the precision and recall curve as in [WTF08, TFW08], precision in a particular hamming radius(we use 3). We also show the training time and compress time (log based). Another criteria we use is the MAP. MAP is defined as follows: average precision at radius r is defined as the division of total number of retrieved good pairs to the total number of retrieved pairs. MAP is defined as the average precision of all hamming radius, from 1 to a max hamming radius we define, could be the length of the hash code, could be the maximal hamming distance of all pairs. Here, we use the hashing code as the maximal hamming radius to calculate. For example, if we calculate hashing code equals to 16. Then this MAP is the mean of average precision from hamming radius 1 to hamming radius 16. We also calculated the area under precision and recall curve for integrated dataset.

3.3 Small dataset

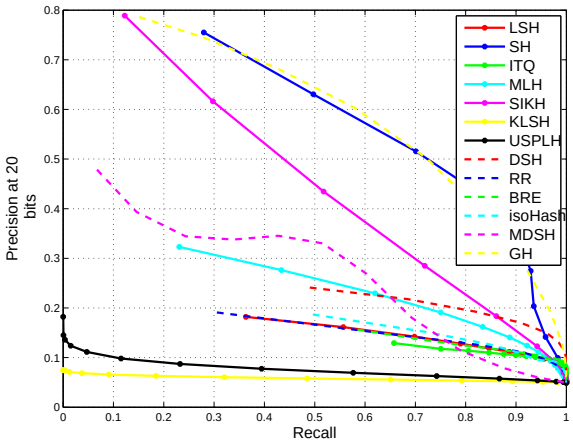
First, I try some small sample datasets. Small sample datasets contain 5000 samples from Labelme, MNIST, Notredame, we name them Labelme5k, MNIST5k, Notredame5k. We also generate 5000 random data from uniform distribution with dimensionality 512, which we call it 'Random Data'. We use 4000 as training data and 1000 as queries. I did two rolls in this section. First, I choose (ℓ_2@5%), then I choose (ℓ_2@10%). Through these two different settings, we can compare different properties of those algorithms and see their pros and cons using different measurement scales. For small dataset, we test number of bits in $10, 20, 30, 40, 50, 60$ bits.



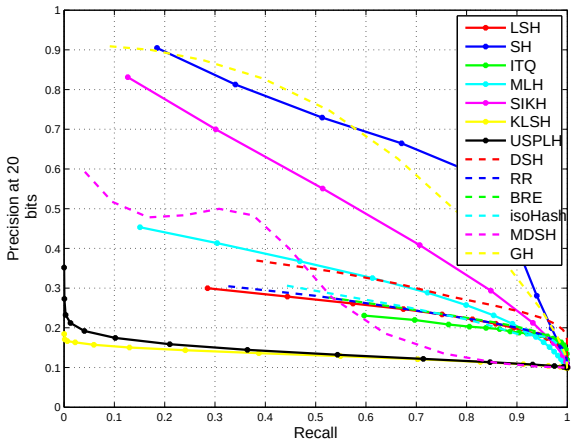
3.1.a: 10 bits



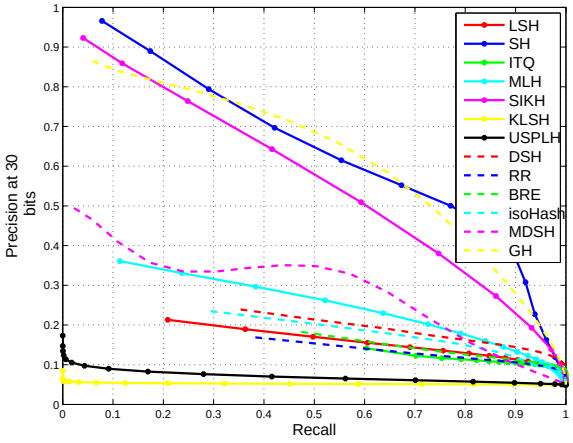
3.1.b: 10 bits



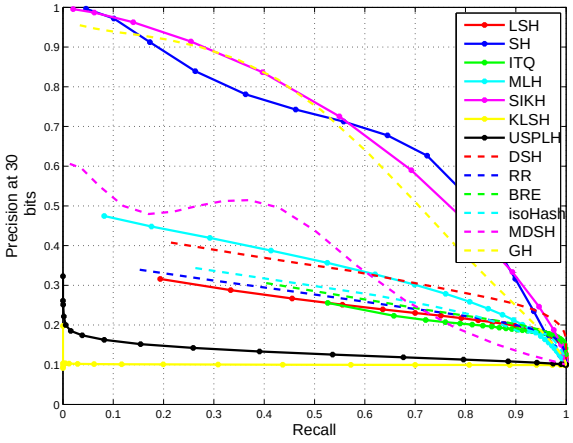
3.1.c: 20 bits



3.1.d: 20 bits

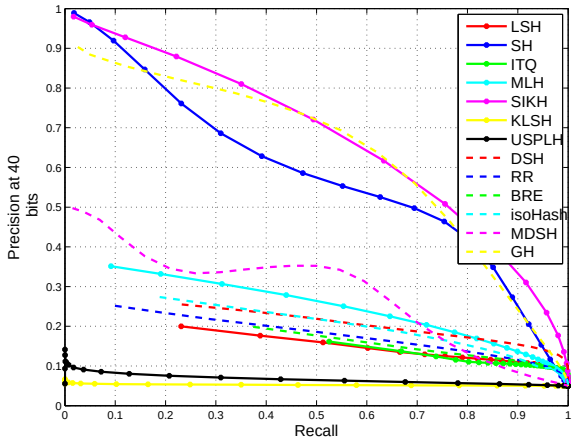


3.1.e: 30 bits

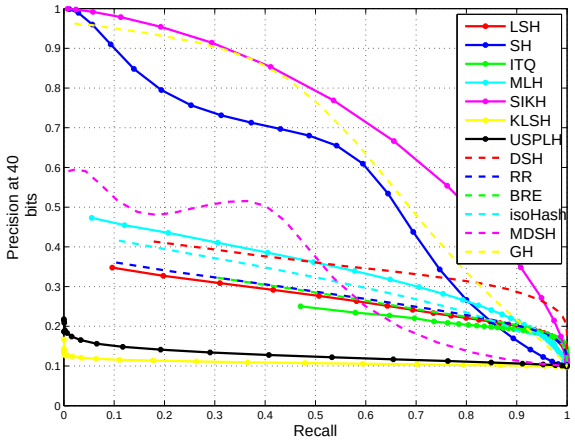


3.1.f: 30 bits

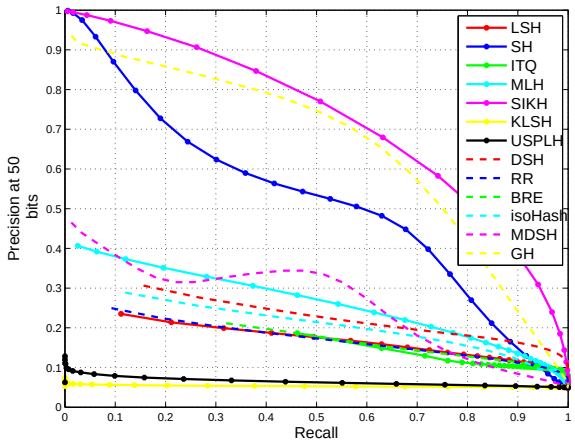
Figure 3.1: Random Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10% .



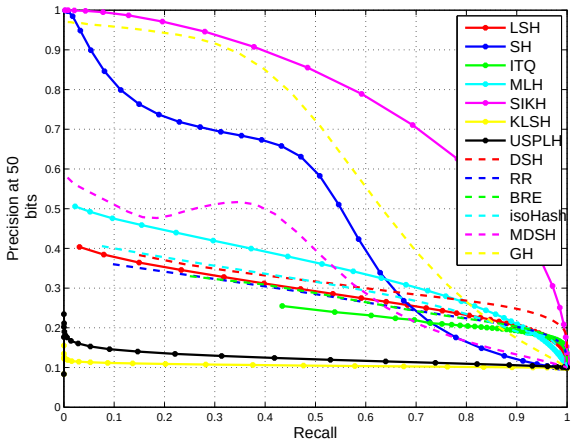
3.2.a: 40 bits



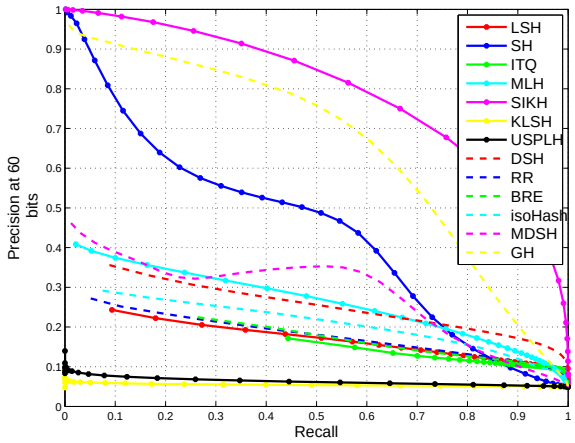
3.2.b: 40 bits



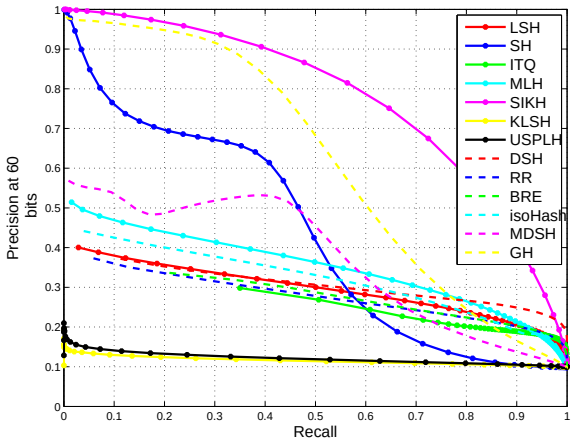
3.2.c: 50 bits



3.2.d: 50 bits

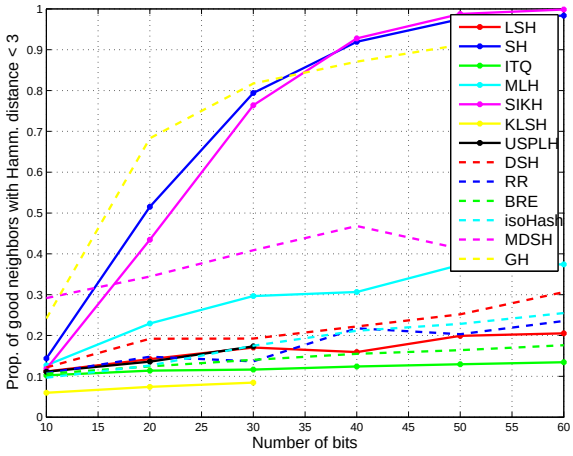


3.2.e: 60 bits

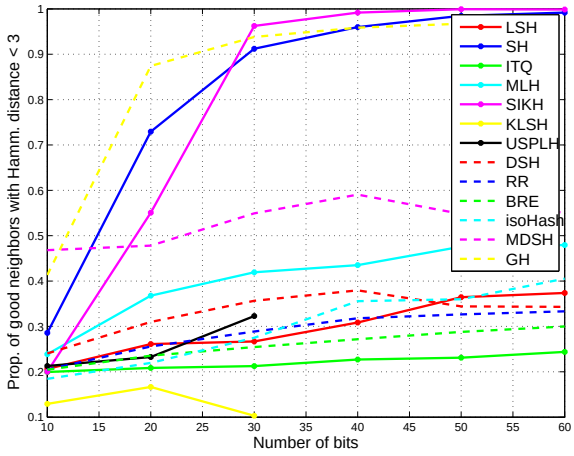


3.2.f: 60 bits

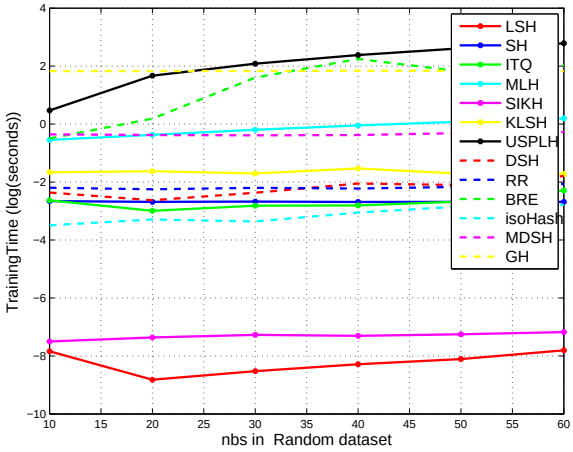
Figure 3.2: Random Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10% .



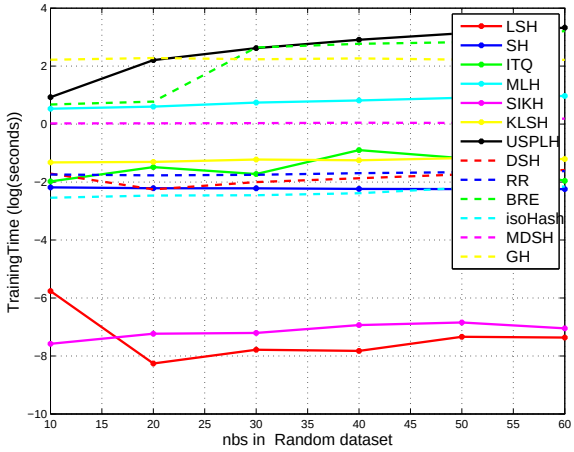
3.3.a: Precision in Hamming radius 3



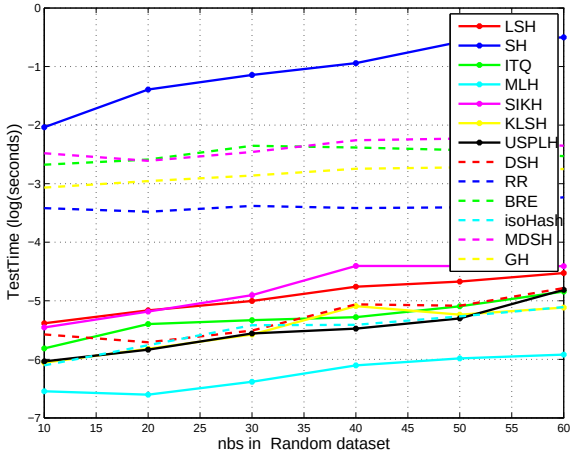
3.3.b: Precision in Hamming radius 3



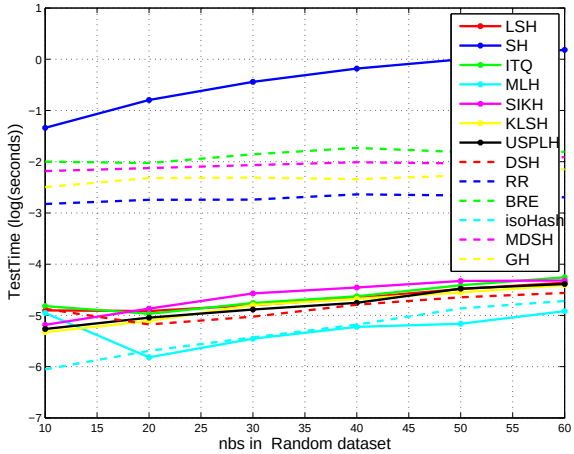
3.3.c: Training Time (log)



3.3.d: Training Time (log)



3.3.e: Compress Time (log)



3.3.f: Compress Time (log)

Figure 3.3: Random Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10%.

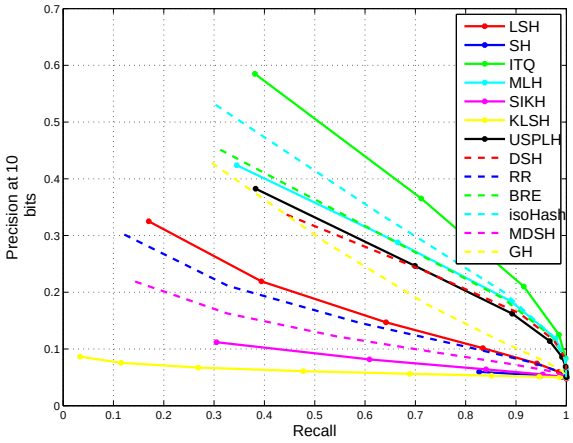
Table 3.1: MAP in Random generated dataset

Methods	10	20	30	40	50	60
LSH	0.09 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00
SH	0.11 ± 0.01	0.20 ± 0.01	0.25 ± 0.01	0.28 ± 0.00	0.29 ± 0.00	0.29 ± 0.00
ITQ	0.09 ± 0.00	0.09 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00
MLH	0.10 ± 0.00	0.12 ± 0.00	0.13 ± 0.00	0.12 ± 0.00	0.14 ± 0.00	0.13 ± 0.00
SIKH	0.11 ± 0.01	0.16 ± 0.02	0.20 ± 0.01	0.23 ± 0.03	0.25 ± 0.01	0.28 ± 0.01
KLSH	0.06 ± 0.00	0.06 ± 0.01	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00
USPLH	0.09 ± 0.00	0.08 ± 0.00	0.08 ± 0.01	0.07 ± 0.01	0.06 ± 0.00	0.06 ± 0.00
DSH	0.10 ± 0.00	0.11 ± 0.01	0.11 ± 0.01	0.11 ± 0.01	0.11 ± 0.01	0.12 ± 0.00
RR	0.09 ± 0.00	0.10 ± 0.00	0.09 ± 0.00	0.11 ± 0.01	0.10 ± 0.00	0.11 ± 0.00
BRE	0.09 ± 0.00	0.09 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00
isoHash	0.09 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.01	0.11 ± 0.00	0.11 ± 0.00
MDSH	**0.15 ± 0.00**	0.17 ± 0.00	0.18 ± 0.00	0.18 ± 0.00	0.16 ± 0.00	0.17 ± 0.01
GH	**0.15 ± 0.00**	**0.25 ± 0.01**	**0.29 ± 0.01**	**0.32 ± 0.01**	**0.35 ± 0.00**	**0.36 ± 0.00**

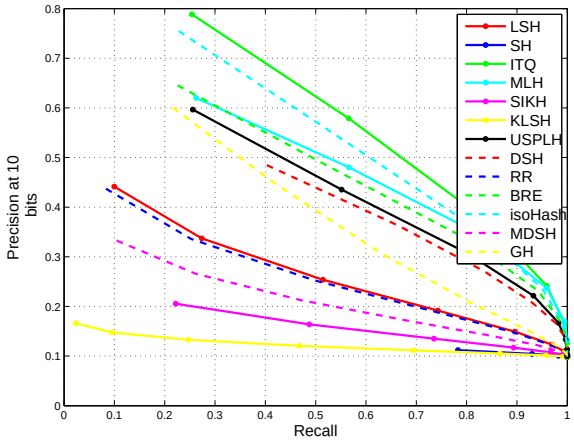
(a) MAP result of Random Dataset at ℓ_2@5%

Methods	10	20	30	40	50	60
LSH	0.17 ± 0.00	0.18 ± 0.01	0.18 ± 0.00	0.19 ± 0.01	0.20 ± 0.01	0.20 ± 0.00
SH	0.21 ± 0.02	0.31 ± 0.01	0.35 ± 0.00	0.36 ± 0.00	0.36 ± 0.00	0.36 ± 0.00
ITQ	0.17 ± 0.00	0.18 ± 0.00	0.18 ± 0.00	0.18 ± 0.00	0.18 ± 0.00	0.18 ± 0.00
MLH	0.18 ± 0.00	0.21 ± 0.00	0.21 ± 0.00	0.21 ± 0.00	0.22 ± 0.00	0.22 ± 0.00
SIKH	0.17 ± 0.01	0.23 ± 0.02	0.31 ± 0.01	0.32 ± 0.01	0.34 ± 0.03	0.34 ± 0.03
KLSH	0.12 ± 0.01	0.13 ± 0.02	0.10 ± 0.00	0.11 ± 0.01	0.10 ± 0.01	0.11 ± 0.00
USPLH	0.16 ± 0.00	0.15 ± 0.00	0.14 ± 0.00	0.13 ± 0.01	0.12 ± 0.00	0.12 ± 0.01
DSH	0.18 ± 0.00	0.19 ± 0.01	0.20 ± 0.01	0.21 ± 0.00	0.20 ± 0.01	0.20 ± 0.01
RR	0.17 ± 0.00	0.18 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.01	0.19 ± 0.00
BRE	0.17 ± 0.00	0.18 ± 0.00	0.18 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.19 ± 0.00
isoHash	0.17 ± 0.00	0.18 ± 0.00	0.18 ± 0.00	0.19 ± 0.00	0.19 ± 0.00	0.20 ± 0.00
MDSH	0.25 ± 0.00	0.25 ± 0.00	0.27 ± 0.00	0.27 ± 0.00	0.26 ± 0.01	0.26 ± 0.01
GH	**0.26 ± 0.00**	**0.36** ± 0.00	**0.39** ± 0.01	**0.41 ± 0.01**	**0.43 ± 0.01**	**0.44 ± 0.01**

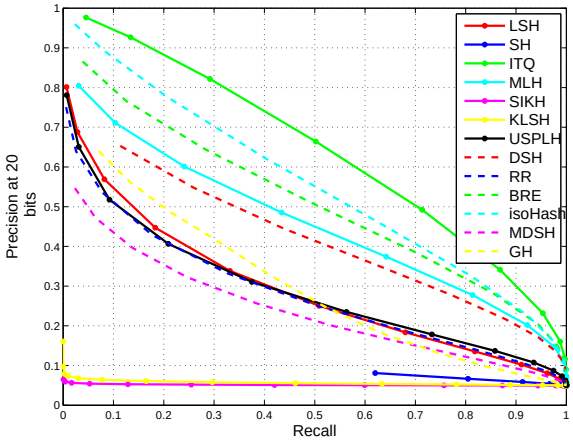
(b) MAP result of Random Dataset at ℓ_2@10%



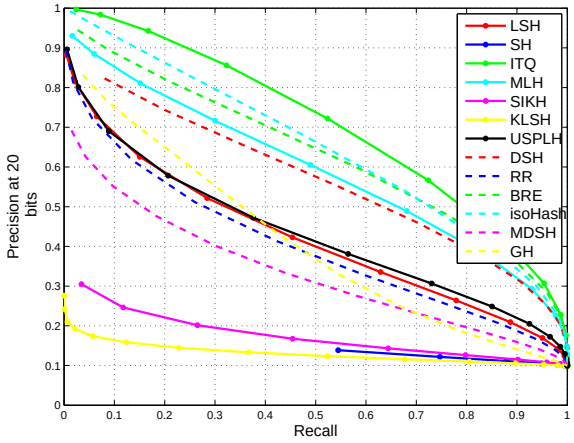
3.4.a: 10 bits



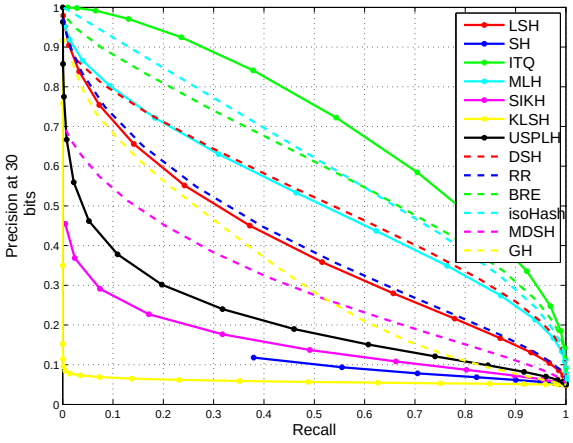
3.4.b: 10 bits



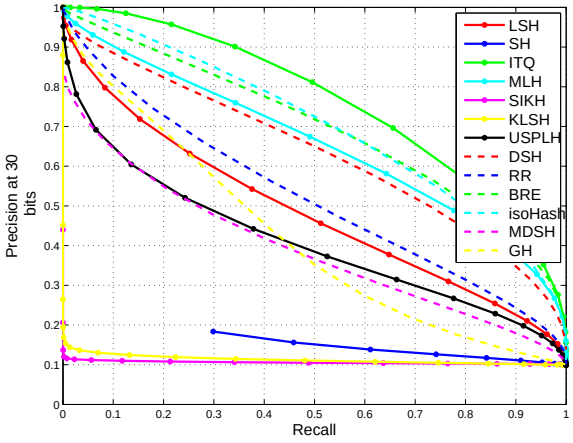
3.4.c: 20 bits



3.4.d: 20 bits

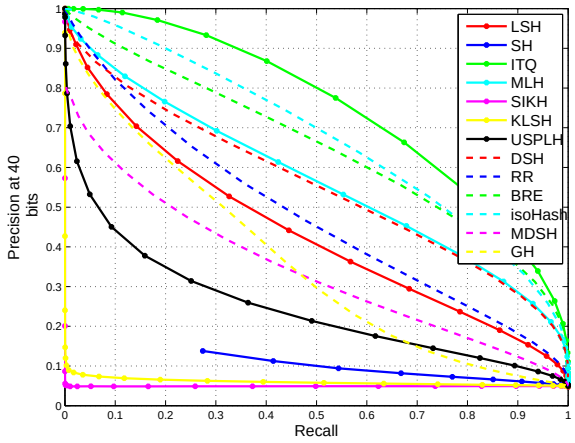


3.4.e: 30 bits

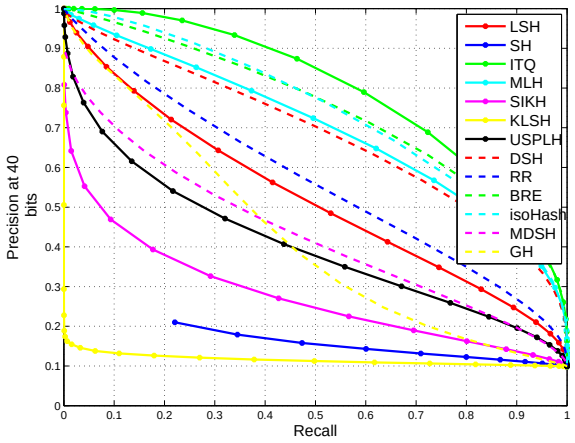


3.4.f: 30 bits

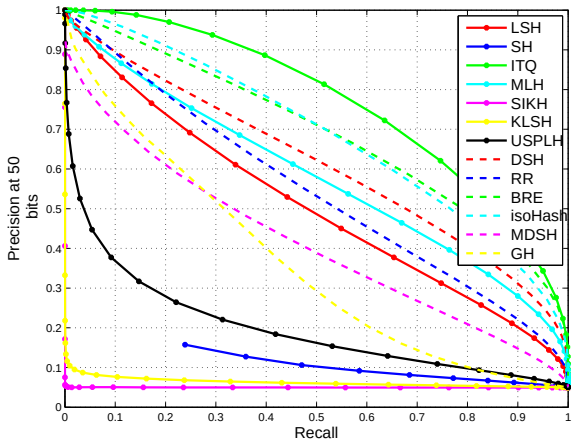
Figure 3.4: Labelme5k Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10% .



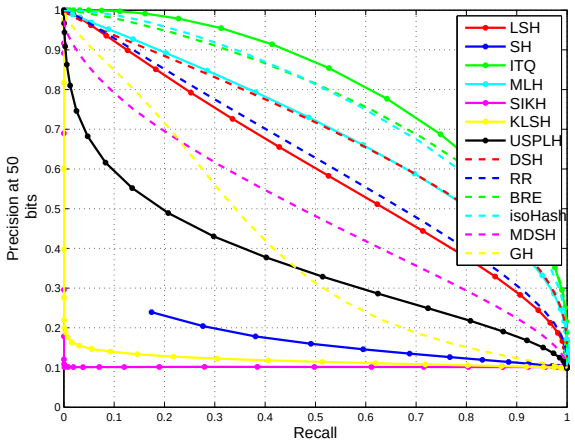
3.5.a: 40 bits



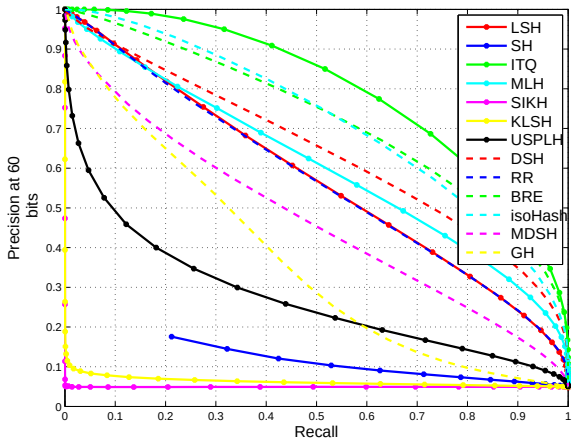
3.5.b: 40 bits



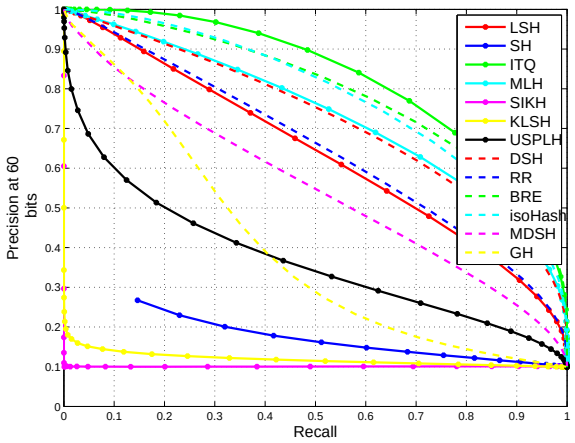
3.5.c: 50 bits



3.5.d: 50 bits

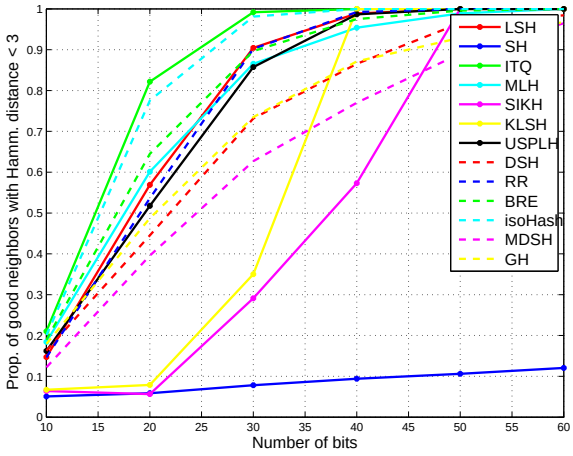


3.5.e: 60 bits

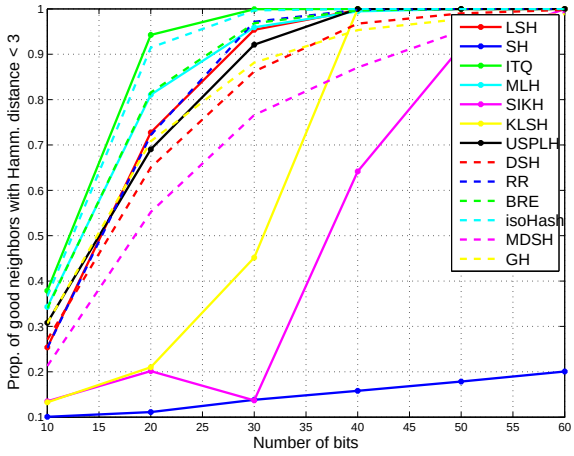


3.5.f: 60 bits

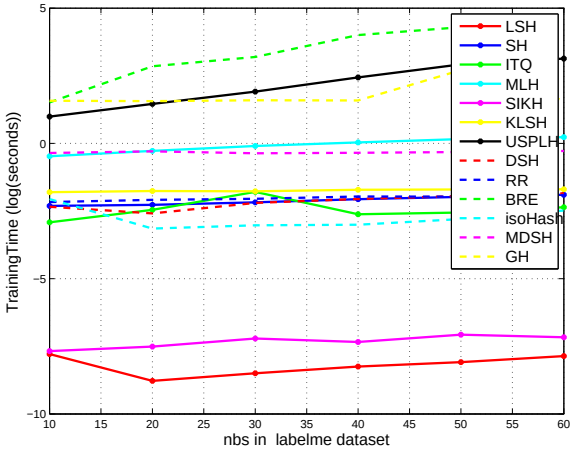
Figure 3.5: Labelme5k Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10% .



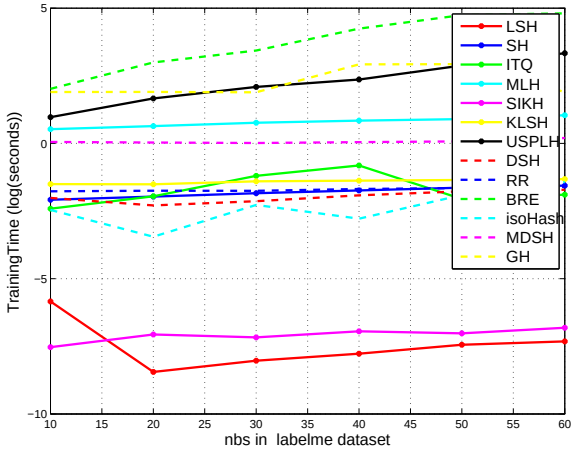
3.6.a: Precision in Hamming radius 3



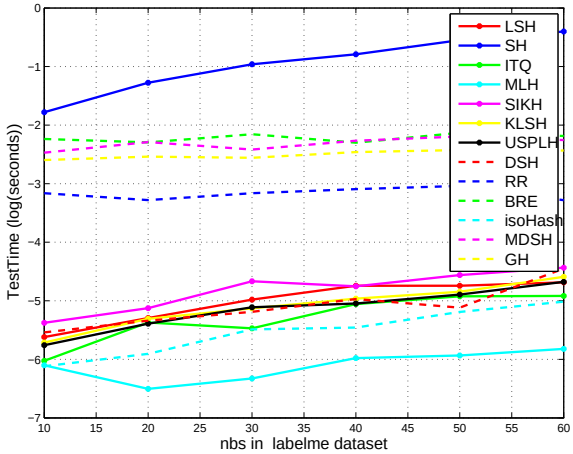
3.6.b: Precision in Hamming radius 3



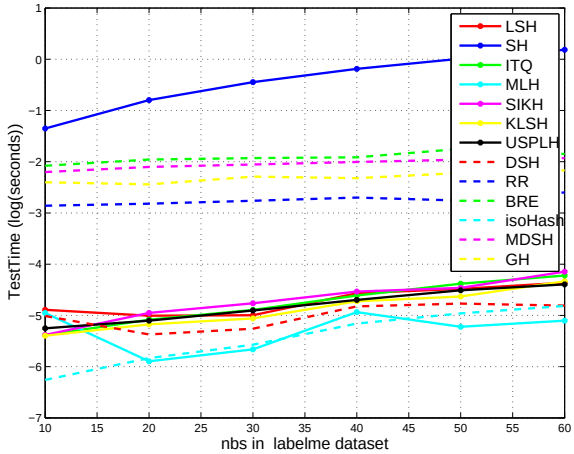
3.6.c: Training Time (log)



3.6.d: Training Time (log)



3.6.e: Compress Time (log)



3.6.f: Compress Time (log)

Figure 3.6: Labelme5k Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10%.

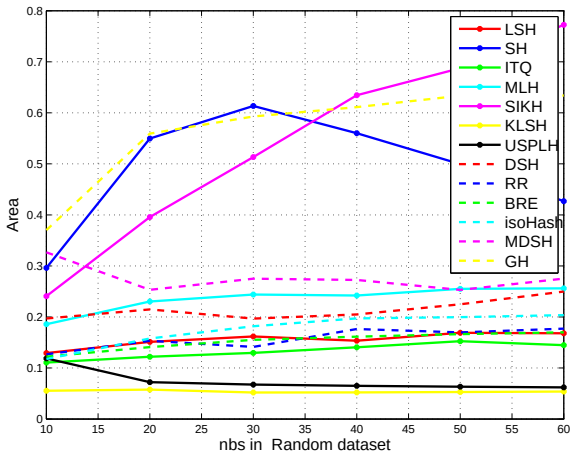
Table 3.2: MAP in Labelme 5k dataset

Methods	10	20	30	40	50	60
LSH	0.11 ± 0.01	0.21 ± 0.00	0.27 ± 0.01	0.31 ± 0.00	0.33 ± 0.01	0.35 ± 0.00
SH	0.05 ± 0.00	0.05 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.00
ITQ	**0.16 ± 0.00**	0.27 ± 0.00	**0.31 ± 0.00**	**0.34 ± 0.00**	**0.36 ± 0.00**	**0.37 ± 0.00**
MLH	0.14 ± 0.00	0.22 ± 0.00	0.27 ± 0.00	0.29 ± 0.01	0.32 ± 0.01	0.33 ± 0.00
SIKH	0.06 ± 0.01	0.05 ± 0.00	0.10 ± 0.01	0.11 ± 0.00	0.15 ± 0.01	0.17 ± 0.01
KLSH	0.06 ± 0.00	0.06 ± 0.00	0.12 ± 0.01	0.19 ± 0.01	0.23 ± 0.00	0.25 ± 0.01
USPLH	0.13 ± 0.00	0.20 ± 0.02	0.25 ± 0.01	0.29 ± 0.01	0.32 ± 0.00	0.33 ± 0.01
DSH	0.12 ± 0.00	0.18 ± 0.01	0.23 ± 0.01	0.25 ± 0.00	0.28 ± 0.01	0.29 ± 0.01
RR	0.11 ± 0.00	0.20 ± 0.01	0.27 ± 0.01	0.32 ± 0.00	0.34 ± 0.00	0.36 ± 0.00
BRE	0.14 ± 0.01	0.23 ± 0.00	0.27 ± 0.00	0.30 ± 0.01	0.31 ± 0.01	0.33 ± 0.01
isoHash	0.15 ± 0.01	**0.26 ± 0.01**	**0.31 ± 0.00**	0.33 ± 0.00	0.35 ± 0.01	0.36 ± 0.00
MDSH	0.09 ± 0.00	0.16 ± 0.01	0.21 ± 0.01	0.24 ± 0.01	0.28 ± 0.01	0.30 ± 0.01
GH	0.14 ± 0.00	0.18 ± 0.00	0.24 ± 0.01	0.28 ± 0.00	0.30 ± 0.01	0.31 ± 0.00

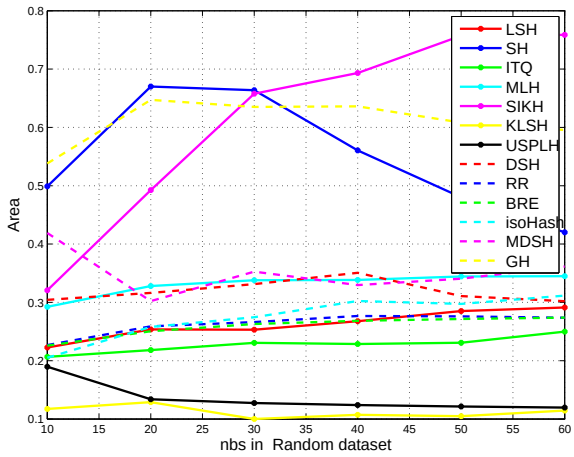
(a) MAP result of Labelme5k Dataset at ℓ_2@5%

Methods	10	20	30	40	50	60
LSH	0.19 ± 0.01	0.30 ± 0.01	0.36 ± 0.01	0.39 ± 0.00	0.41 ± 0.01	0.43 ± 0.00
SH	0.10 ± 0.00	0.10 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00	0.11 ± 0.00
ITQ	**0.27 ± 0.00**	0.36 ± 0.00	**0.40 ± 0.00**	**0.42 ± 0.00**	**0.43 ± 0.00**	**0.44 ± 0.00**
MLH	0.24 ± 0.01	0.33 ± 0.00	0.37 ± 0.00	0.39 ± 0.00	0.41 ± 0.00	0.42 ± 0.01
SIKH	0.12 ± 0.01	0.13 ± 0.02	0.12 ± 0.01	0.20 ± 0.04	0.19 ± 0.00	0.20 ± 0.01
KLSH	0.12 ± 0.00	0.14 ± 0.00	0.19 ± 0.01	0.23 ± 0.01	0.27 ± 0.00	0.30 ± 0.01
USPLH	0.23 ± 0.01	0.29 ± 0.02	0.33 ± 0.01	0.37 ± 0.01	0.39 ± 0.00	0.39 ± 0.01
DSH	0.21 ± 0.01	0.28 ± 0.01	0.32 ± 0.02	0.36 ± 0.00	0.37 ± 0.01	0.38 ± 0.01
RR	0.19 ± 0.01	0.30 ± 0.01	0.37 ± 0.00	0.40 ± 0.00	0.42 ± 0.00	0.43 ± 0.00
BRE	0.24 ± 0.01	0.33 ± 0.00	0.37 ± 0.00	0.39 ± 0.00	0.41 ± 0.00	0.41 ± 0.00
isoHash	0.26 ± 0.00	**0.36 ± 0.00**	**0.40 ± 0.00**	**0.42 ± 0.00**	**0.43 ± 0.00**	**0.44 ± 0.00**
MDSH	0.17 ± 0.00	0.25 ± 0.00	0.30 ± 0.00	0.33 ± 0.00	0.37 ± 0.00	0.39 ± 0.00
GH	0.22 ± 0.00	0.29 ± 0.00	0.34 ± 0.00	0.37 ± 0.01	0.38 ± 0.01	0.39 ± 0.00

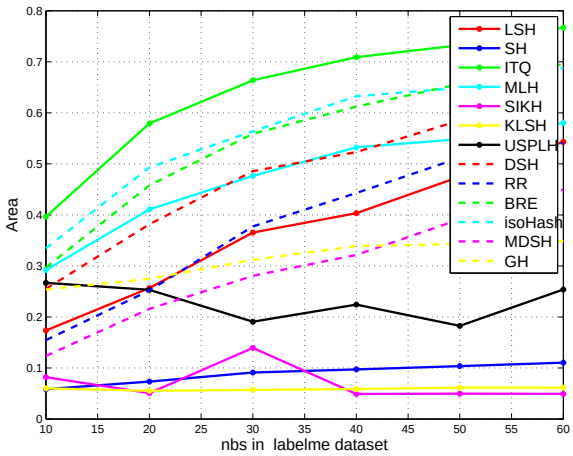
(b) MAP result of Labelme5k Dataset at ℓ_2@10%



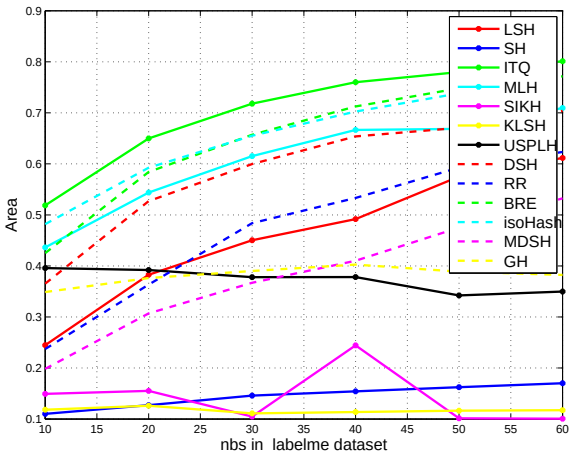
3.7.a: Random Dataset



3.7.b: Random Dataset



3.7.c: Labelme5k



3.7.d: Labelme5k

Figure 3.7: Area Under P-R Curve, left column is ℓ_2@5%, right column is ℓ_2@10% .

Here, we only take random dataset(figure 3.1 to 3.3 and table 3.1) and Labelme5k dataset (figure 3.4 to 3.6 and table 3.2) as examples to save spaces. MNIST5k(figure A.1 to A.3 and table A.1) and Notredame5k dataset (figure A.4 to A.6 and table A.2) are shown in appendix.

From small sample dataset, we can see that: SIKH, SH, GH perform well in Random Dataset. Especially SIKH. It achieves much better results in the precision and recall curve. But when it arrives to the real world dataset, like in other dataset, SIKH performs much worse.

In the real world dataset, ITQ and isoHash perform much better. See Labelme 5k for example. ITQ achieves the best in precision and recall curve. isoHash, BRE, MLH, DSH also performs well. We can also see that MDSH performs much better than SH, which is the advantage of MDSH compared with SH.

Then let's see the time cost. For training time, LSH and SIKH are relatively low. BRE, GH, USPLH, MLH are relatively high. For compressing time, MLH, isoHash, MDSH are lower than other methods. SH, BRE, MDSH are relatively higher than other methods.

Then we see the precision in Hamming radius 3. As the number of bits increases, the trends of this precision is increasing. But in the random generated dataset, we can see that SH and SIKH perform much better than other methods. Especially SH, which performs the worst in labelme 5k dataset. This is because SH assumes a multidimensional uniform distribution generates the data. So in random dataset, this requirement is perfectly met. So SH performs well in random generated dataset. But in real world dataset, this requirement can not be met, many dataset may far away from uniform distribution, so in this case, SH performs much worse.

Compare with left and right columns, we can see that there is little difference between ℓ_2@5% and ℓ_2@10% settings. For the MAP we define, ITQ achieves the best almost all the time except in the random dataset. Similar results appear in Areas between P-R curve 3.7.

Table 3.3: MAP result of Labelme22k Dataset at ℓ_2@10%

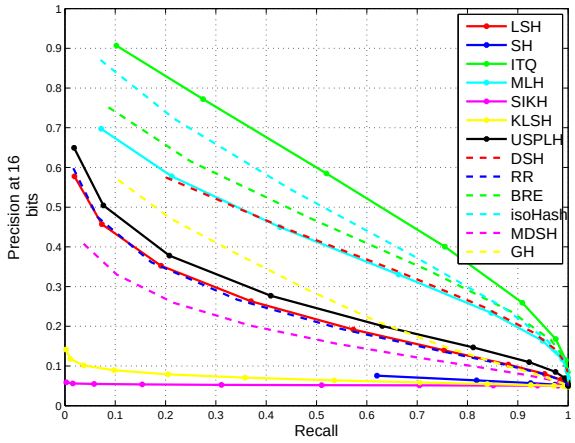
Methods	16	24	32	64	128
LSH	0.16 ± 0.01	0.24 ± 0.01	0.28 ± 0.01	0.36 ± 0.00	0.39 ± 0.00
SH	0.05 ± 0.00	0.05 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.00
ITQ	**0.23 ± 0.00**	**0.29 ± 0.00**	**0.32 ± 0.00**	**0.37 ± 0.00**	0.39 ± 0.00
MLH	0.19 ± 0.00	0.24 ± 0.00	0.28 ± 0.00	0.34 ± 0.00	0.37 ± 0.00
SIKH	0.05 ± 0.00	0.05 ± 0.00	0.08 ± 0.01	0.17 ± 0.01	0.27 ± 0.00
KLSH	0.07 ± 0.00	0.09 ± 0.01	0.13 ± 0.01	0.25 ± 0.01	0.33 ± 0.01
USPLH	0.17 ± 0.01	0.20 ± 0.01	0.26 ± 0.00	0.31 ± 0.02	0.21 ± 0.02
DSH	0.16 ± 0.00	0.19 ± 0.01	0.22 ± 0.02	0.29 ± 0.00	0.32 ± 0.01
RR	0.16 ± 0.01	0.23 ± 0.01	0.29 ± 0.00	0.36 ± 0.01	**0.40 ± 0.00**
BRE	0.20 ± 0.00	0.25 ± 0.01	0.28 ± 0.00	0.33 ± 0.01	0.36 ± 0.00
isoHash	0.22 ± 0.00	0.27 ± 0.00	0.31 ± 0.00	**0.37 ± 0.00**	0.39 ± 0.00
MDSH	0.13 ± 0.00	0.19 ± 0.01	0.21 ± 0.01	0.31 ± 0.00	0.36 ± 0.00
GH	0.17 ± 0.00	0.21 ± 0.01	0.25 ± 0.01	0.32 ± 0.02	0.36 ± 0.00

Table 3.4: MAP result of MNIST60k Dataset at ℓ_2@10%

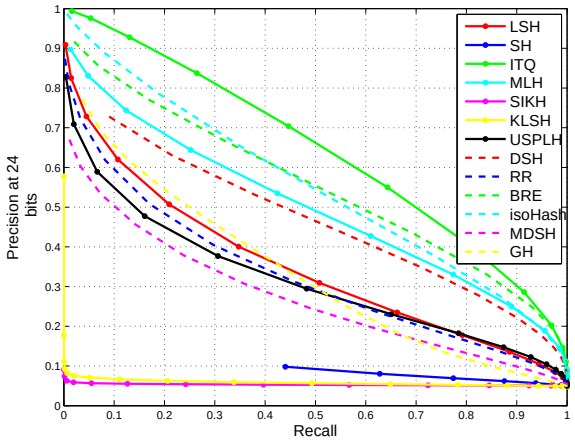
Methods	16	24	32	64	128
LSH	0.25 ± 0.01	0.33 ± 0.01	0.37 ± 0.00	0.44 ± 0.01	0.46 ± 0.00
SH	0.24 ± 0.01	0.30 ± 0.01	0.34 ± 0.00	0.40 ± 0.01	0.42 ± 0.00
ITQ	**0.36 ± 0.00**	**0.40 ± 0.00**	**0.43 ± 0.00**	**0.47 ± 0.00**	**0.49 ± 0.00**
MLH	0.32 ± 0.00	0.37 ± 0.00	0.40 ± 0.00	0.44 ± 0.01	0.46 ± 0.00
SIKH	0.14 ± 0.00	0.14 ± 0.02	0.17 ± 0.03	0.10 ± 0.00	0.10 ± 0.00
KLSH	0.18 ± 0.01	0.26 ± 0.01	0.30 ± 0.01	0.38 ± 0.01	0.42 ± 0.01
USPLH	0.31 ± 0.01	0.37 ± 0.01	0.35 ± 0.00	0.44 ± 0.01	0.43 ± 0.01
DSH	0.27 ± 0.01	0.31 ± 0.01	0.35 ± 0.01	0.40 ± 0.00	0.44 ± 0.01
RR	0.25 ± 0.01	0.33 ± 0.00	0.37 ± 0.01	0.44 ± 0.00	0.46 ± 0.01
BRE	0.32 ± 0.00	0.37 ± 0.00	0.40 ± 0.00	0.44 ± 0.00	0.46 ± 0.00
isoHash	0.34 ± 0.00	0.39 ± 0.00	0.42 ± 0.00	0.46 ± 0.00	0.48 ± 0.00
MDSH	0.26 ± 0.01	0.31 ± 0.01	0.34 ± 0.00	0.40 ± 0.00	0.45 ± 0.00
GH	0.23 ± 0.01	0.29 ± 0.01	0.32 ± 0.01	0.39 ± 0.01	0.44 ± 0.00

3.4 Integrated Dataset

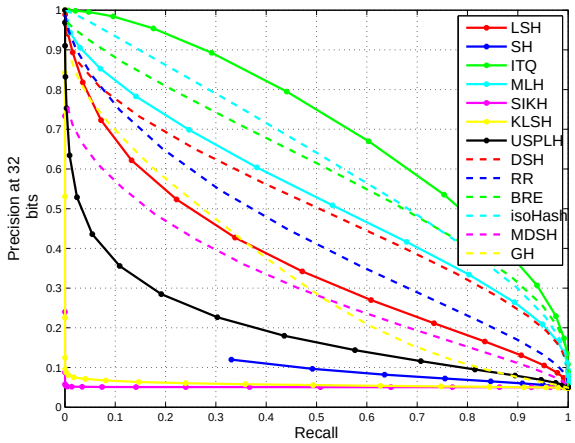
For whole dataset test, we use MNIST, Labelme, Caltech101, CIFAR10. MNIST contains 60000 training data and 10000 test data. Labelme contains 20000 training data and 2019 test data. Caltech101 contains 6677 training data and 2000 test data. CIFAR10 contains 50000 training data and 10000 test data. In all the training data, I randomly split 20% of the training data as validation data. All parameters are choose by cross validation, please refer to Parameter section.. For large dataset, we test number of bits in $16, 24, 32, 64, 128$ bits For all Integrated Dataset experiment, we choose ℓ_2@10%



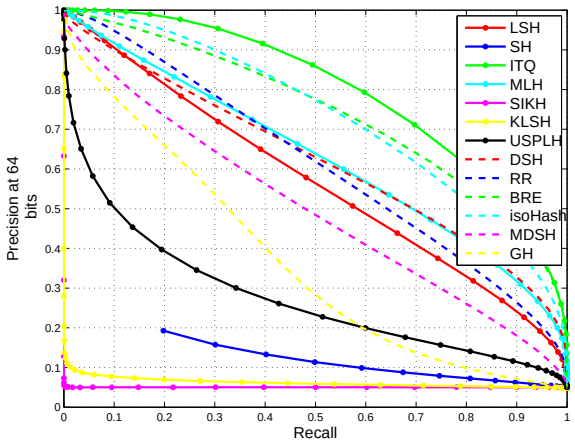
3.8.a: 16 bits



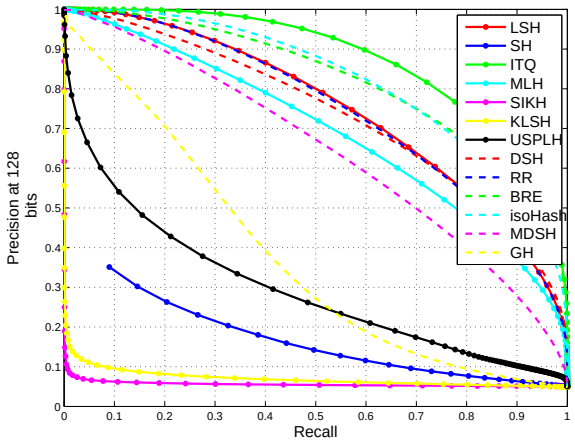
3.8.b: 24 bits



3.8.c: 32 bits

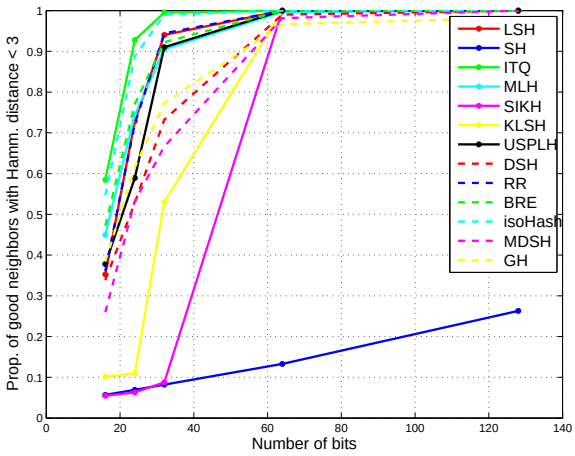


3.8.d: 64 bits

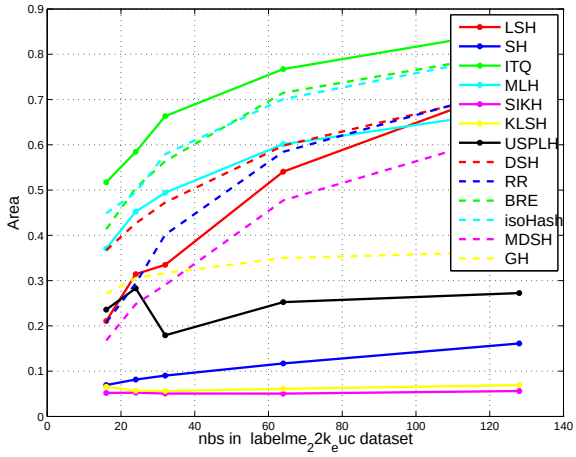


3.8.e: 128 bits

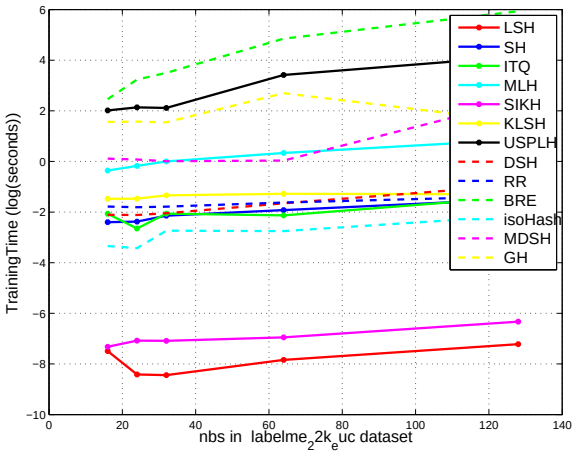
Figure 3.8: Labelme22k Data Precision and Recall Curve at ℓ_2@10% .



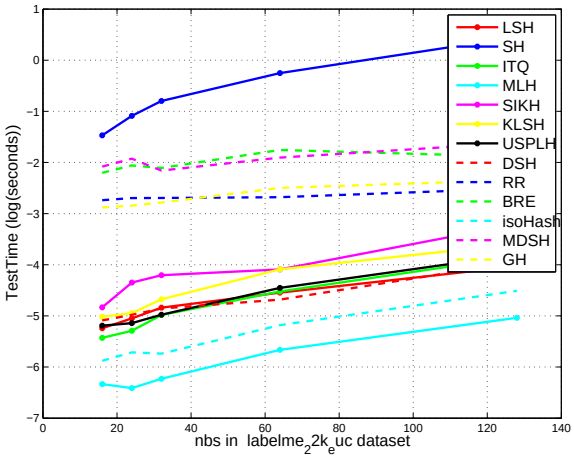
3.9.a: Precision in Hamming radius 3



3.9.b: Area under P-R curve

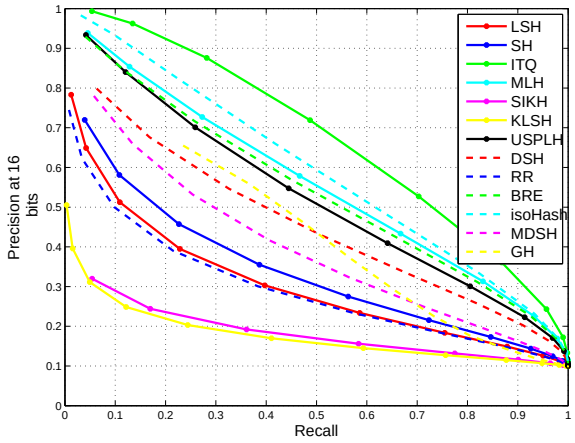


3.9.c: Training Time (log)

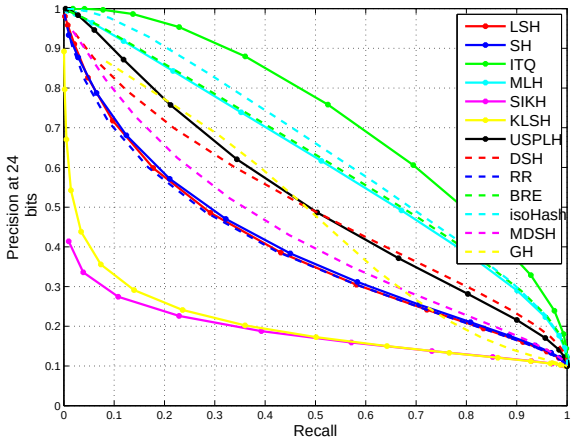


3.9.d: Compress Time (log)

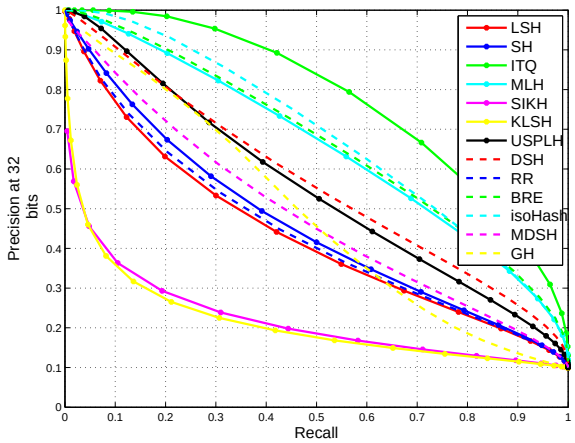
Figure 3.9: Labelme22k Data at ℓ_2@10%.



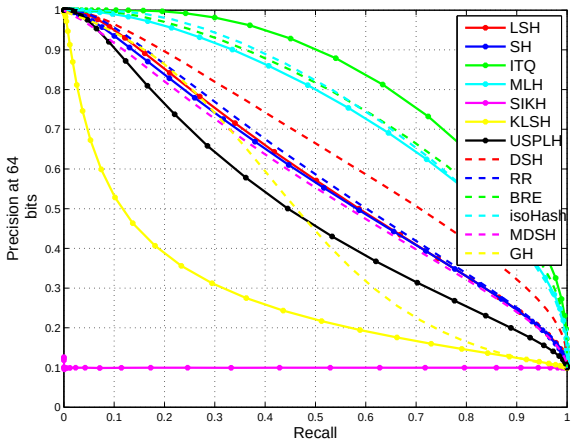
3.10.a: 16 bits



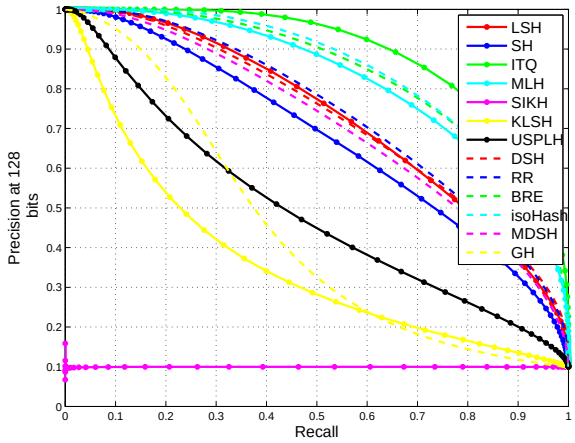
3.10.b: 24 bits



3.10.c: 32 bits

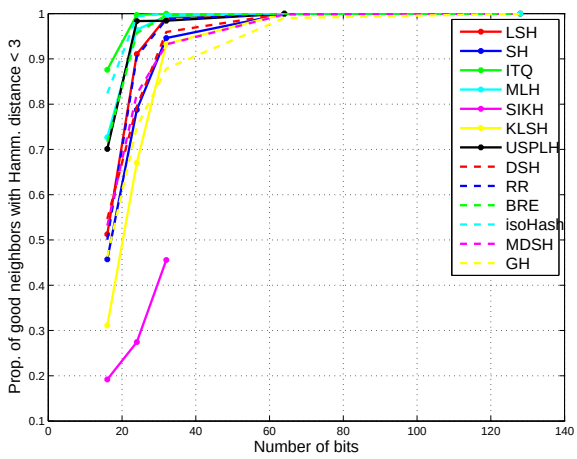


3.10.d: 64 bits

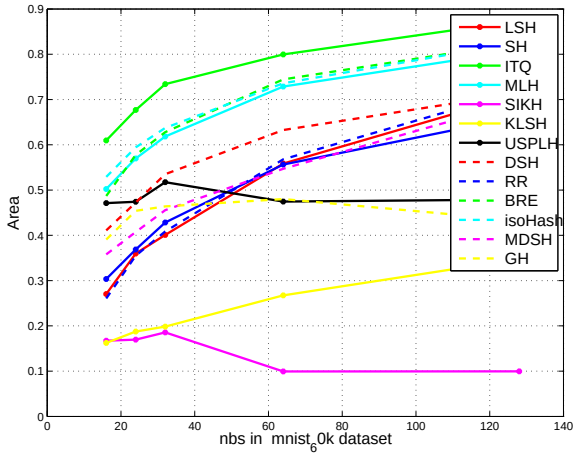


3.10.e: 128 bits

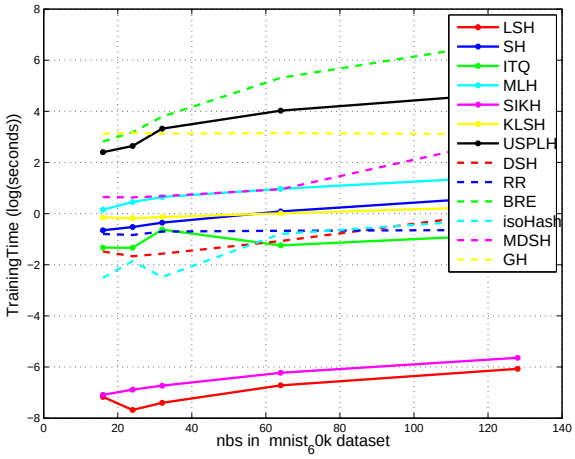
Figure 3.10: MNIST60k Data Precision and Recall Curve at ℓ_2@10% .



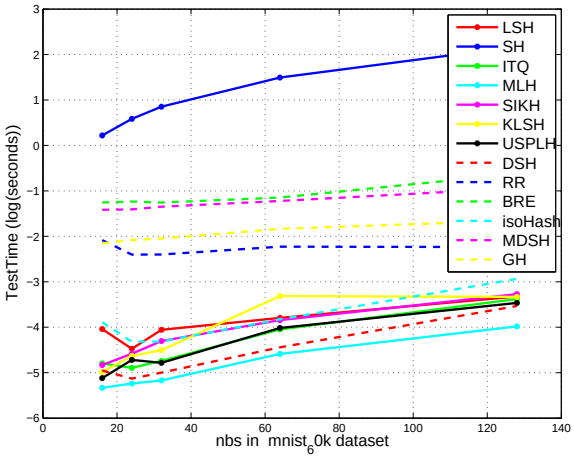
3.11.a: Precision in Hamming radius 3



3.11.b: Area under P-R curve



3.11.c: Training Time (log)



3.11.d: Compress Time (log)

Figure 3.11: MNIST60k Data at ℓ_2@10%.

In integrated datasets, we take Labelme 22k and MNIST 60k for examples to discuss. I put CIFAR10 and Caltech101 dataset results in appendix. Since ℓ_2@5% and ℓ_2@10% are similar in sampled dataset, so we use ℓ_2@10% for integrated dataset.

ITQ again achieves almost all of the best results in this section. It performs well in both small number of bit and large number of bit, see figure 3.8. In Labelme dataset, BRE, isoHash, MLH also perform well. SIKH, SH, USPLH, KLSH performs a little worse. But in MNIST 60k dataset, USPLH, SH, DSH, KLSH performs much better than in labelme dataset. This indicated that many methods are very related to the dataset we use, different test ground can get different results. Especially methods like SH that have an assumption on data distribution. So in reality, if we need to use one method on a particular dataset, we may need to do good test before we pick one hashing algorithm.

Training time and compressing time are similar with the sampled dataset. Area under P-R curve is more obvious than the P-R curve. As the number of bits increases, the treads are increasing in figure 3.9.b and 3.11.b. But SIKH performs worse compared with other methods. ITQ achieves the best in all 13 methods.

As a conclusion, we can see that ITQ is of the best value on euclidean search task among all the 13 hashing methods we listed above. 1), It performs good on all evaluation criteria and all dataset except the random generated dataset that sampled from uniform distribution. 2), the training time and test time is quite small. 3). It has very few parameters to tune, which makes ITQ very simple and easy to implement. Although from the training and test time table, we may see that many methods cost relatively similar time, but some methods will use a lot of time for cross validation, such as MLH. MLH has a lot more parameters to tune, which costs cross validation a long time to finish. ITQ is easy and of the best value. If I can only choose one for Euclidean search task, I will definitely choose ITQ among the 13 methods we compared.

Chapter 4

Cross View Hashing using Deep learning

4.1 Introduction

In the era of big data today, massive data can be easily collected in real-world data mining applications by various tools from heterogeneous sources, especially in emerging social media and multimedia domains. The advent of big data has presented various open challenges of developing fast retrieval techniques to facilitate emerging big data analysis and mining tasks. An emerging challenge is the (approximate) Cross-View Nearest Neighbor Search (CVNNS), where the feature space (view) of the query instance can be different or completely diverse from that of the target instances to be retrieved in the database. This is a very hard problem, which is almost of no hope to solve if no training data or prior knowledge is available; even when training data is available, it remains a very challenging open problem for developing an effective learning and searching scheme that can bridge the gap between the two views by mining the training data.

There are many (approximate) nearest neighbor search algorithms we mentioned in previous chapter on single-view high-dimensional data and they are well studied. But few algorithms was proposed for cross view data. The key challenge in the cross-view nearest neighbor search task is that the input spaces (views) of the query instance and the target instances to be retrieved can be diverse or completely different, which prevents the straightforward application of the existing learning to hash algorithms to solve the CVNNS tasks. To attack this challenge, one direction is to develop some technique that can directly map data instances from different input spaces to the same binary (hamming) space. This approach usually has to involve very complicated optimization tasks [KU11] and cannot take advantages of many successful learning to hash algorithms for single-view data. Unlike this approach, in this paper, we present a new deep learning to hash scheme for cross-view nearest neighbor search, which explores the emerging deep learning

techniques in learning a shared representation by mining training data to bridge the gap between the query view and the target view and is able to take advantages of the existing learning to hash techniques for performing hashing on the shared representation. We conduct an extensive set of experiments on varied settings of cross-view image retrieval tasks, in which the encouraging results show that the proposed method outperforms the state-of-the-art approaches.

The key difficulty of cross-view learning to hash is to find a shared representation which is able to close the representation gap between the query and the target instances. We propose a new cross-view learning to hash framework by exploring the emerging deep learning techniques. In particular, we investigate deep learning architectures based on Restricted Boltzmann Machines (RBMs) in learning shared representation for cross-view learning to hash, and extensively evaluate their empirical performance of nearest neighbor search under several challenging settings.

4.2 Cross-View NN Search.

For many real-world data mining applications, it is critically important to develop cross-view Nearest Neighbor (NN) search techniques, particularly for multimedia search and mining or cross-lingual information retrieval tasks [UK10]. In literature, some hash-based approaches have been proposed in attempting to address the related tasks of cross-view NN search [KU11, QL11]. The first kind of approach is the cross-view similar search (CVSS) study in [KU11], which formulated the cross view learning to hash as an optimization task, similar to spectral hashing, in order to construct hash functions that can minimize all the distances between the two views of training instances. The principle behind this approach is to explore the idea of Canonical Correlation Analysis (CCA) [HSSSt07] for learning the shared representation. However, the key limitation of these work is that their shallow architectures can only capture the linear relationship among the two views/spaces which clearly would not be powerful enough to model the complex relationship across different views.

In this paper, we aim to explore deep learning techniques to overcome this limitation.

4.2.1 Sparse Multi-Modal Hashing

Recently, a related work (SMMH) [WYY$^+$14] has been proposed. They also try to learn hash codes and retrieve across modalities. They obtain sparse codesets for the data objects via joint multi-modal dictionary learning. Basically, their work have four steps as follows: 1 Construct a so-called Hypergraph [ZLW$^+$11] to get the intra-modality similarity and inter-modality similarity between the same modality and different modalities. 2 Construct two dictionaries jointly on both modalities using the hypergraph constructed.

Here, they share the same idea of sparse coding and try to use a small number of corresponding 'basis atoms' to represent data objects. 3 Get the sparse coefficients for a data object. Here, they first define the code length. According to code length, get the corresponding dimensional indices with the largest absolute values. 4 Use the sparse coefficients of each data object to generate its compact representation and use this to do hash.

4.2.1.1 Comparison Between SMMH and Learning to Hash Community

SMMH brings new ideas into the learning to hash community. I will do a qualitative discussion in this section to compare with SMMH and traditional methods. We can see, it dose have some differences.

How They Work: There is a huge difference between this method and traditional learning to hash work. Traditional methods learn a compact representation to represent one object then through some methods(like binarization) to get binary hash codes. But in this method, hash codes are the dimension indices with largest values in all dimensions. For example, if we want to learn a hash code of length 30. For traditional method, we only need to learn a representation of objects with dimension 30, then get the hash code by simply binarization. But with this method, if the sparsity of sparse coding is set to 0.1, we must learn at least 300 dimensional representation of objects, then choose the indices of top 30 with the largest values to be its hash codes.

Time Complexity: SMMH consists of an off-line multimodal dictionary learning and on-line sparse codeset learning [WYY+14]. The off-line multimodal dictionary learning is solved by iterative manner, and the time complexity is $O((p_x + p_y)K^2 + p_xKN_x + p_yKN_y)$, where p_x is the dimension of modality x, p_y is dimension of modality y. K is the number of instances in dictionary. N_x is number of objects in modality x and N_y is the number of objects in modality y. The time complexity of on line sparse codeset learning is $O(p_xK)$. A bottleneck of this work, which the author didn't mention, is that the whole framework is based on hypergraph. Constructing such a hypergraph is actually very time consuming. The time complexity is roughly $O(N^{2d})$ where N is the number of training samples and d is the dimensionality. Large scale training will make the problem intractable.

Evaluation Criteria: There exists a controversial problem in the experiment of SMMH, which makes it different with the learning to hash community. The author uses his own evaluation criteria to compare with other method traditional methods. The way he use is to set the size of dictionary and the sparsity to mimic the representation power of traditional hash code. The reason he gives is that both this kind of representation and the traditional learning to hash representation have the same level of representation power. For example, they set dictionary size 50 and set sparsity 0.2, that is to choose 10

coefficient from 50 and he assumes it is 'almost' the same level as the traditional hash code representation power with code length 32. They neither define 'approximation level' here nor proof that it is OK to do so. From my point of view, I can not agree with this. We can see, they did experiment using 32 bits with $C(50, 10)$, 48 bits with$C(100, 10)$, 64 bits with $C(150, 15)$. First, they should not do evaluation based on similar (in fact not very similar) representation power, they should do evaluation with the same hash codes, because in all the learning to hash work in literature, no one use approximation of representation power to compare hash methods. In the literature, every work did evaluation based on same hash codes. Hash algorithms are very sensitive to the length of hash codes, a small change in hash codes (like 8 bits and 12 bits) with have difference that can not be ignored. So this kind of comparison should be very accurate. Second, even if they can use approximation, they should use a good approximation. We see one example, 64 bits, representation power is 2^{64}. While $C(150, 15)$ is 8.8 times larger than 2^{64}. The author treat them have the same representation power and compare together. It is very likely that SMMH is better because they choose a combination of dictionary size and sparsity, which has a 8.8 times larger representation power than traditional methods as shown in their experiment.

Conclusion: Although SMMH is not the first work using Sparse coding in hashing [CMP12], The idea of SMMH is new, the joint dictionary learning algorithm is also novel. But I think, some steps(construction of hypergraph) need future improvement, and the theory (using different criteria to compare between their work and traditional ones) needs further proof, to make it more practical on real word application and be accepted by the learning to hash community.

4.3 Deep Learning to Hash for Cross-View NN Search

In this section, we present a deep learning to hash framework for cross-view NN search, which is illustrated in Figure 4.2. In the following, we will first formulate the problem and then present the key deep learning architectures for learning to hash towards cross-view NN search.

4.3.1 Problem Formulation and Overview

Let us denote by $\{o_i = (x_i^{(1)}, x_i^{(2)})\}_{i=1}^n$ a collection of two-view data instances, and $x_i^{(k)} \in \mathbb{R}^{d_k}$ the k-th view of the object o_i, where d_k is the dimensionality of the k-th view and $k = 1$ or 2. Let us denote by z_i the actual observation of the object o_i, which consists of any subset of the two views of object o_i. The goal of our cross-view learning to hash is to search for a mapping function as follows:

$$f : z_i \mapsto \{-1, 1\}^d$$

where d is the number of bits for the final output binary codes. As a special case, if z_i always consists of all the two views of object o_i, a straightforward approach to solving the problem is to treat the entire two-view data as a single view and then apply a conventional single-view learning to hash algorithm to generate the hash codes.

The key challenge of cross-view learning to hash is to search for the mapping function f which must ensure the output binary codes $f(z_i)$ are always highly similar, if not identical, no matter how is the subset of views observed in z_i. One possible way to solve it is to formulate this task as a single learning to hash optimization problem [KU11], and attempt to solve it directly. Such approach however has two major limitations. First of all, it is often not easy to directly solve the complicated optimization problem. Second, it cannot take advantages of many existing learning to hash algorithms which are widely and successfully used for searching single-view data.

In this work, we propose a two-stage deep learning to hash scheme for cross-view NN search, which can be formulated as follows:

$$f = g \circ h : z_i \mapsto \{-1, 1\}^d \tag{4.1}$$

$$g : z_i \mapsto \mathbb{R}^{d'}, h : \mathbb{R}^{d'} \mapsto \{-1, 1\}^d \tag{4.2}$$

where g is a mapping function which maps diverse z_i to the same d'-dimensional space, and h does the mapping from d'-dimensional space to d-dimensional Hamming space.

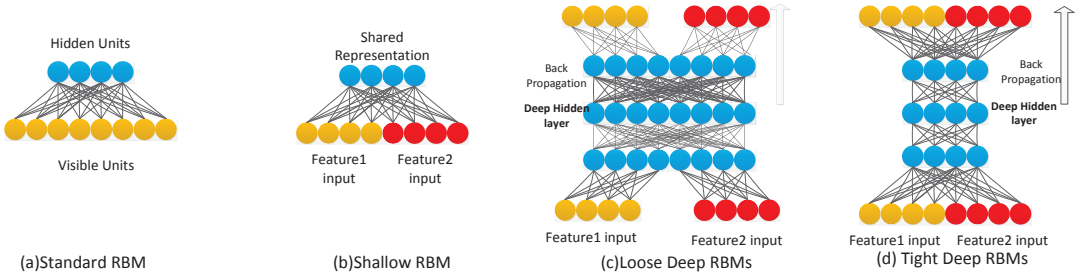


Figure 4.1: Illustration of four different RBM models as cross-view learning to hash architectures: (a) is a standard RBM model; (b) is a Shallow RBM model, which concatenates two sets of features as input to an RBM; (c) is the 'Loose Deep RBM' model proposed in [NKK+11]; and (d) is our proposed 'Tight Deep RBM' model which involves tight connections. A set of yellow units stands for one feature set, a set of red units stands for another feature set, and a set of blue units denotes the representation of the hidden layers. In (c) and (d) models, the back propagation part is included, the second hidden layer is the deep hidden layer, in which we will learn a shared representation.

Clearly, one can easily develop a good hashing function h using any existing learning to hash algorithms. In our empirical study, we adopt LSH and SH. The key challenge of

the above two-stage learning to hash is to find the mapping function g which essentially maps any subset of multiple views to the same shared representation space. In the following, we present a deep learning approach to tackling this challenge. For simplicity, in the following, we will illustrate the proposed scheme using only two views. Note that the similar idea can be easily generalized to the cases of multiple views.

4.3.2 Overview of Cross-View Deep Learning Architectures.

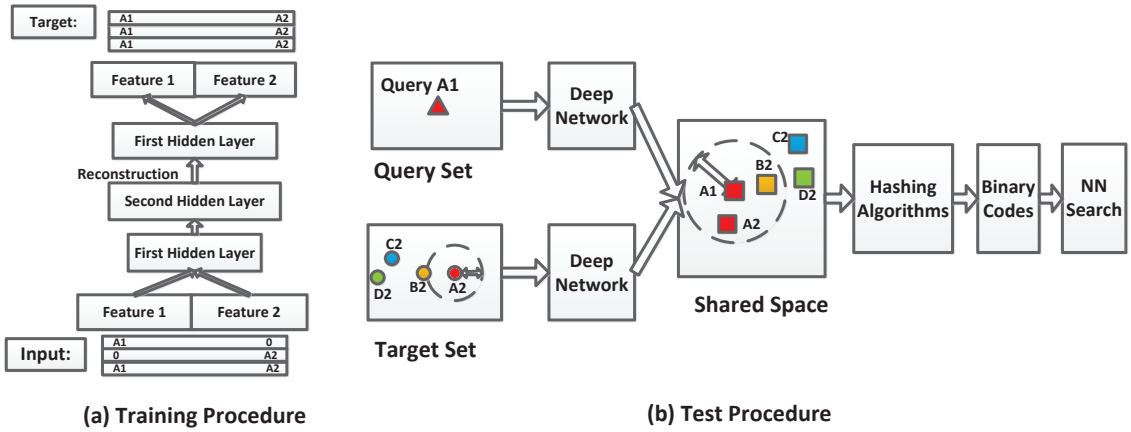


Figure 4.2: Framework of CVNNS via Deep Learning to Hash: (a) Training Procedure: the input to the model is an augmented vision of two views (A1 and A2) with additional examples that have only a single-view as input ([A1,0] and [0,A2]), but require the network to reconstruct both views ([A1,A2]); (b) Test Procedure: when only one view is provided as test data, we project test query data instances and target candidate instances into the second hidden layer, and then perform cross view hashing algorithm (LSH or SH) on this shared space. Here, triangle and circle stand for two views. The squares in the shared space stand for projections of different instances from different views. The same color stands for the same data. (We draw one test query A1 in the Query Set for concise illustration). Using one view (A1) as query, we want to find another modality (A2) or its neighbor (A2' neighbor B2 in this example) based on some similarity metric selections.

In our framework, a cross-view deep learning to hash task consists of two major phases: (i) training phase that aims to learn the function f consisting of two components: the cross-view mapping function g and the hashing function h; and (ii) retrieval/test phase for cross-view NN search where the views of the query and the target instances in database can be diverse or completely different. We assume that the two views are both available for learning the function f during the training phase. In this paper, we mainly focus on learning the cross-view mapping function g and adopt either LSH or SH as the hashing function h. In the following, we will discuss the design of several different learning architectures to address the learning task of our framework.

4.3.3 Shallow Learning Architectures.

Figure 4.1 (a) and (b) show the design of shallow learning architectures for the cross-view learning task. Figure 4.1 (a) shows the **Standard RBM (RBM)**, which trains a single-layer RBM for each single view, and cannot solve the cross-view NN search problem because the two views are completely decoupled. Figure (b) shows the **Shallow RBM (SRBM)** approach, which is the simplest approach that trains a single-layer RBM by combining two views as input for a standard RBM. However, the SRBM approach may not be powerful enough to learn the shared representation of both input data.

4.3.4 Deep Learning Architectures.

Here, we investigate two different designs of deep learning architectures for cross-view learning to hash, as shown in Figure 4.1 (c) and (d), respectively. We discuss the details of the two architectures as follows.

Loose Deep RBMs (LDRBM): As shown in Figure 4.1 (c), the input features of both views are *loosely* connected at the beginning for learning the shared representation. Specifically, at the beginning, features of each view are input to a separate RBM for learning a representation; further, the outputs of their individual representations are concatenated as the input for another higher layer RBM for learning the shared representation (the second hidden layer). The training process has two phases: pre-training and fine-tuning. During the pre-training phase (Figure 4.2 (a)), the input to the whole learning architecture is an augmented set which consists of training examples with the combined two-view features and some additional examples that consist of only a single-view feature. During the fine-tuning phase, the learning goal is to require the deep learning network to effectively reconstruct both modalities of the input data. We note that this architecture, to some extend, shares the same inspirit of the multimodal deep learning architecture proposed in [NKK$^+$11] when considering only two views. The potential disadvantage of this learning architecture is that the loose connections of the input features from different views may not be able to fully capture the interactions among all the features from the beginning, which motivates to explore the following alternative.

Tight Deep RBMs (TDRBM): As shown in Figure 4.1 (d), the input features of both views are tightly connected between visible and first hidden layers at the beginning in order to fully explore the interactions between the two kinds of input features. In particular, the TDRBM model essentially learns a two-layer RBM network, i.e, the visible layer and the first hidden layer form one SRBM (which we adopt the Gaussian-Bernoulli RBM for the first layer in our approach), and the first and second hidden layers form another RBM. By such approach, using the same number of layers as the LDRBM network, we can capture the shared representation through two hidden layers (while LDRBM captures the share representation only for the second hidden layer). As

a result, the final shared representation learned by TDRBM would have a better chance to capture more in-depth interactions between the two views. For the training process, we adopt the similar strategy as the LDRBM.

4.3.5 Pros and Cons Between SMMH and CVNNS

Construction of hypergraph in SMMH [WYY+14] is very time consuming. The complexity is roughly $O(N^{2d})$ where N is the number of training samples and d is the dimensionality. Large scale training will make the problem intractable. Although deep learning is well known of its high time consuming training procedure, we have various training algorithms to help reduce the training complexity, like batch training, contrastive divergence learning.etc, (please refer to Chapter 2). These algorithms are very practical in real world applications. So our method is much better on this comparison. But from Chapter 2, we can see that [LWKC11] also needs to construct an Anchor Graph. It actually share some similar ideas with SMMH. It may be an alternative solution to the hypergraph construction in SMMH.

Another advantage of our method compared with SMMH is that we belong to one of the traditional learning to hash community. So directly comparison between other methods in the community with our method is very simple and easy. But if SMMH wants to do comparison with other methods, it may need to find the combination between dictionary size and sparsity. There are so many such combinations, each one has different result, which means SMMH will need a lot of time and effort on choosing combination instead of randomly picking such a pair that has an approximate representation power with traditional hash codes. This also makes it impractical in the real world applications.

Another advantage of our method compared with SMMH is that deep learning theories we use here, including approximation algorithms, have been well proved as shown in Chapter 2. But some theories in SMMH need future proof. SMMH has some default assumption, which has no evidence that they are true, nor have they ever been used by the learning to hash community.

SMMH has some advantages. One advantage of SMMH over our method is that the usage of sparse coding is novel, although they are not the first one using it in hashing [CMP12], it brings new ideas and new algorithms in the learning to hash community. Another advantage is that they use a good application on image and text retrieval, which is very practical and commonly seen in the real world. I also mention this before in Future Work Section. Since our work is much earlier than their work, so it is reasonable that they have a latest application in the paper. While our work only focus on image retrieval between different features, as I have mentioned, more novel applications will appear in our Future Work.

4.4 EXPERIMENTS

4.4.1 Overview.

In this section, we conduct empirical study through three extensive sets of experiments for image retrieval tasks: (i) single view NN Search, (ii) cross-view NN search on two-view data, and (iii) cross-view NN search on multi-view data. The first experiment aims to examine the advantages of exploiting different views of data in the training stage, and evaluate if and how deep learning improves the learning to hash performance. The other two experiments aim to show the efficacy and advantages of the proposed deep learning to hash technique for cross-view NN search in comparison to the state-of-the-art approaches.

4.4.2 Experimental Testbed.

We adopt three public image datasets: **ImageCLEF**, **Caltech101** [FFFP06] and **NUS-Wide-Lite** [CTH+09]. **ImageCLEF** consists of 7157 medical images in 20 classes for benchmark image retrieval competition. **Caltech101** consists of 8677 images in 101 classes. We extract five different sets of global visual features in a total of 809 dimensions for both **ImageCLEF** and **Caltech101** datasets, including 81 Color features, 120 Gabor features, 512 Gist features, 59 LBP features, and 37 Edge features, which are generated by the "FELIB" feature extraction toolbox [ZHLY08]. For performance metric, we adopt the precision-recall curve and mean average precision (MAP), which have been widely used in image retrieval studies [TFW08, KD, WTF08]. **NUS-Wide-Lite** consists of $55,615$ images, in which $22,807$ images are used for training and the other images are used for test/retrieval. The dataset provides six kinds of visual features, including 64-D Color histogram, 144-D Color correlogram, 225-D block-wise Color moments, 73-D Edge direction histogram, 128-D Wavelet texture and 500-D Bag of words. There are 81 concepts in total and each image has a few concepts [1].

4.4.3 Experimental Settings.

In the following experiments, we evaluate the performance of our models under different datasets and varied settings to examine every aspect. Below we discuss our experimental settings in detail.

Configuration of Deep Models: (i) Units type: We use Gaussian visible units that are connected to the input data. When training a deeper layer, we use binary visible

[1] NUS-Wide-Lite dataset is available online, which can be downloaded from http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm

units. (ii) Number of units: The number of visible units is equal to the dimensionality of its input, thus in different experiments, the number of visible units is different. Hidden layer structure is an overcomplete style. For Color-Gabor, Gist-Full cross view hashing experiment (including Color-Gabor single view NN search experiment), the number of hidden layer features is 2×overcomplete for visible units in all layers. For LBP-Gabor, CCC-EW (see Experiment III) cross view hashing experiment, the number of first hidden layer features is 2× over-complete for visible units, and the number of second hidden layer features is equal to (1×overcomplete) the first hidden layer features. These settings had been well justified in theory [HOT06].

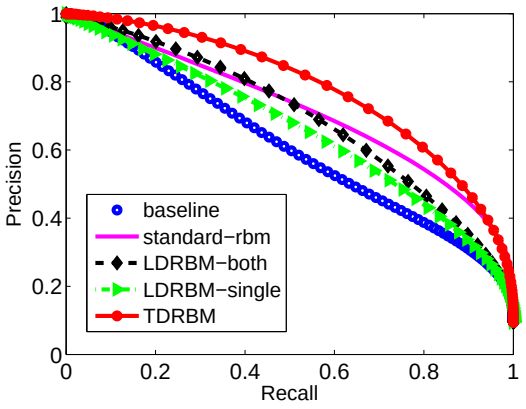
Training Procedures: The model uses greedy layer-wise pre-training with CD equals to 1. We use 200 epoches to train the first Gaussian-Bernoulli RBM and 50 epoches to train the second RBM. In the fine-tuning stage: our objective function is to minimize the square error loss between the inputs and the targets. The method of conjugate gradients is utilized and three line searches are performed in each epoch. We use 10 epoches totally in back-propagation.

Experimental Procedures: In the training phase, both features are provided (Figure 4.2 (a)). In the test phase, only one test modality is provided (assume the test modality is $Modality1$ and the test data is $A1$, as shown in Figure 4.2 (b)). In single view NN search experiment, we use feature $A1$ to find nearest neighbors of $A1$ in $Modality1$, while in cross view hashing experiment: we use feature $A1$ to find nearest neighbors of $A2$ in $Modality2$. For LDRBM and TDRBM, the training and test procedures are shown in Figure 2 (a) and (b), respectively. All the tests run 10 times (except CCC-EW cross-view hashing that runs only 3 times due to the high time cost on evaluation [WTF08, TFW08]). All the results reported in our experiments are the average results.

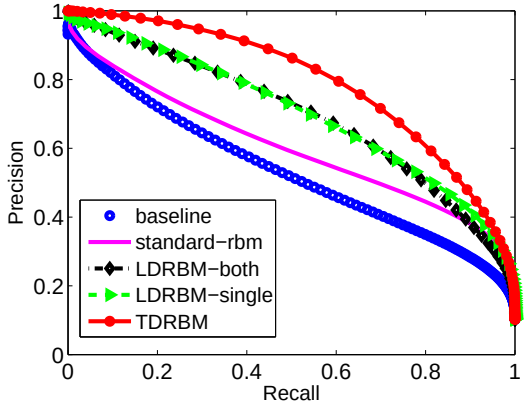
Definitions of "Relevance" Ground Truth: We adopt two kinds of "relevance" definitions for the ground truth: (i) Euclidean distance ($\ell_2@n\%$): top $n\%$ nearest neighbors of the query instance retrieved using the exact Euclidean distance on the target view will be treated as the ground truth relevant images in the retrieval task [WTF08, TFW08]; (ii) Semantic similarity: a returned image is defined relevant to the query only if it has the same semantics as the query (e.g., they belong to the same class or share the same tags).

Size of Training and Test Sets: For all the following experiments (except CCC-EW test in Experiment III), we randomly draw 2000 data points as queries. To train the deep models, we randomly draw 2000 data instances from the rest instances in the data set. For the CCC-EW settings, the details will be discussed in Experiment III.

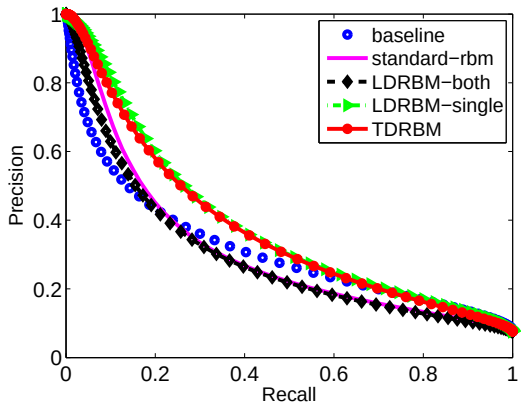
4.4.4 Experiment I: Single View NN Search



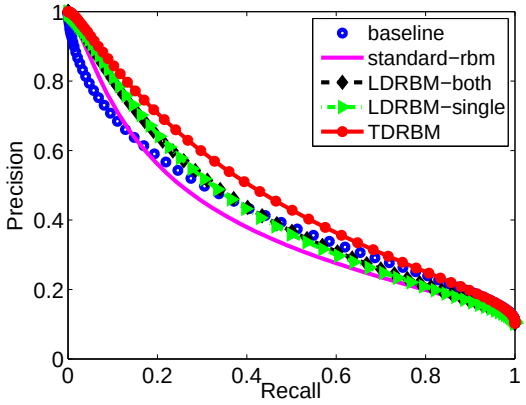
4.3.a: Color feature search by LSH



4.3.b: Gabor feature search by LSH



4.3.c: Color feature search by SH



4.3.d: Gabor feature search by SH

Figure 4.3: Single View NN Search using 256-bit hash codes on the data sets of ImageCLEF and Caltech101: (a) and (b) are the results of ImageCLEF, (c) and (d) are the results of Caltech101.

This experiment aims to examine if training using two views can improve the hashing tasks by learning the shared representation. We thus compare the following learning to hash algorithms: (i) **Baseline**: to adopt the original features for the hashing task;(ii) **Standard RBM** (Figure 4.1 (a)): it first projects data into the hidden representation space and then performs hashing in that space, which only uses one view for training and test; (iii) **LDRBM-single** (Figure 4.1 (c)) is a variant of LDRBM, which assumes biases of RBMs are zeros, similar to [NKK+11]; (iv) **LDRBM-both** (Figure 4.1 (c)) does not omit the biases in the hidden units, which not only takes one feature as input to the

model, but also fills in another view with zeros as inputs; and finally (v) the **TDRBM** model.

We define the relevance ground truth using the Euclidean distance (ℓ_2@5%) criteria in this experiment. We choose "Color" and "Gabor" features on **ImageCLEF** and **Caltech101** datasets to evaluate the performance, respectively. Figure 4.3 shows the set of evaluation results on the NN search using the 256-bit hash codes generated by the above compared algorithms. We draw several observations from the results. First of all, the standard RBM does not always perform well, and sometimes even performs worse than the **Baseline** (e.g., Figure 4.3 (d)). In contrast, the other three models outperform the baseline for most cases (except in Figure 4.3(c) and (d) where LDRBM-both seems slightly worse than the baseline). Among all the algorithms, TDRBM always achieves the best results for most cases. This result indicates that learning the shared representation from training data of more than one views is beneficial to facilitating the learning to hash tasks.

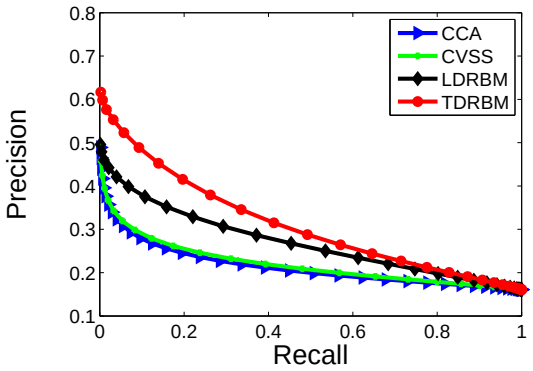
256-bit hash codes are used to strengthen the variation of performance. For each hashing method and each feature as test feature in the experiment, we show a representative result in Figure 4.3. From Figure 4.3, the Standard RBM does not perform well. It performs even worse than using original feature for hashing directly. For other three models: LDRBM-single, LDRBM-both and TDRBM , they all perform better than baseline on the first two cases(Figure 4.3(a) and (b)). But, in Figure 4.3(c) and (d), LDRBM-both is no better than standard RBM and baseline, LDRBM-single performs similarly to TDRBM in Figure 4.3(c) case. The proposed TDRBM model performs good among all the cases.

From this experiment, we can prove that deep learning models are beneficial to learning the shared representation by exploiting the multi-view information available in the training phase.

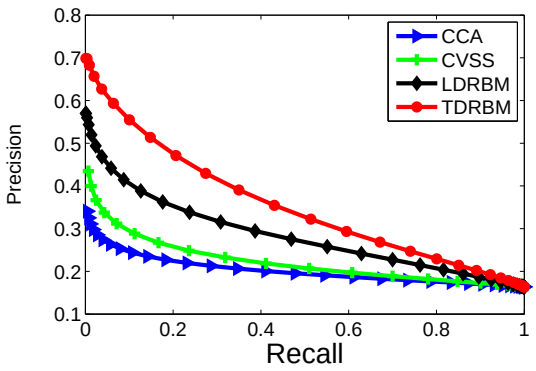
4.4.5 Experiment II: Cross-View NN Search across Two Views

In this section, we evaluate the performance of the proposed deep learning to hash technique for cross-view NN search, which adopts one view of the data to search for data from another view. Specifically, we adopt two pairs of features for evaluation: "LBP-Gabor" and "Color-Gabor" features on the **ImageCLEF** dataset. We evaluate both Semantic search (in LBP-Gabor test) and Euclidean search (in "Color-Gabor" test). For comparison, we choose CCA [HSSSt07] and CVSS [KU11] as two baselines: (i) **CCA**: CCA has been commonly used for multi-view data mining tasks [CKLS09]. Specifically, in the training phase, we use CCA to find the canonical correlation vectors of two features. In the test phase, we transform the query data points and data points in the candidate set

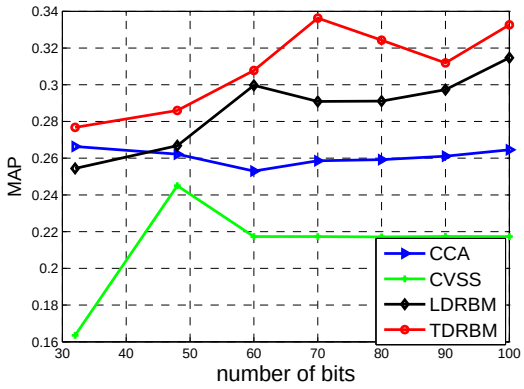
to the same low dimensional CCA embedding space. On the same low dimensional embedding space, we apply a hashing algorithm (LSH or SH) for retrieval. (ii) **CVSS**: the Cross-View Similarity Search algorithm proposed in [KU11], which is a state-of-the-art approach for cross-view hashing in Multilingual search area. Specifically, in the training phase, it learns a transformation matrix; in the test phase, it applies the transformation matrix to obtain hash codes of queries, and then performs the retrieval. CVSS is a different hashing method compared with LSH and SH. We compare CVSS with our model in all settings. As LDRBM-single and LDRBM-both perform similarly, we simply adopt LDRBM-single as the LDRBM model in the following experiments.



4.4.a: From LBP to Gabor search by LSH



4.4.b: From Gabor to LBP search by LSH



4.4.c: From Gabor to LBP search by SH

Figure 4.4: Cross view hashing between LBP and Gabor features with 100-bit hash codes.

LBP-Gabor Semantic Search on "ImageCLEF": In this experiment, we examine two scenarios: (i) use LBP feature as query to find nearest neighbors in the Gabor space, and (ii) vice versa. For each set of 2000 random queries, we evaluate their performance on a variety of hashing settings: 16-bit, 24-bit, 32-bit, 48-bit, 60-bit, 70-bit, 80-bit, 90-bit, 100-bit, and 200-bit hash codes. The average MAP result over 10 runs using LSH is summarized in Table 4.1. The proposed TDRBM always achieves the best performances for varied settings of bit sizes. We also plot the precision and recall curve of 100-bit using LSH in Figure 4.4 (a) and (b). The performance of CVSS is better than that of CCA, while LDRBM is better than these two baselines. Among all, TDRBM always achieves the best performance.

Table 4.1: Evaluation of MAP performance of cross-view hashing on LBP and Gabor features on the ImageCLEF dataset.

Methods	16	24	32	48	60	70	80	90	100	200
CCA	0.1799	0.1869	0.1938	0.1986	0.2016	0.2031	0.2066	0.2145	0.2110	0.2270
CVSS	0.1635	0.1628	0.1696	0.1636	0.2239	0.2240	0.2202	0.2165	0.2147	0.2155
LDRBM	0.1883	0.1957	0.1994	0.2314	0.2323	0.2279	0.2474	0.2584	0.2650	0.2956
TDRBM	**0.2213**	**0.2365**	**0.2469**	**0.2764**	**0.2654**	**0.3131**	**0.2767**	**0.3427**	**0.3022**	**0.3530**

(a) From LBP queries to Gabor search using LSH

Methods	16	24	32	48	60	70	80	90	100	200
CCA	0.1749	0.1801	0.1894	0.1820	0.1900	0.1914	0.1949	0.1982	0.2001	0.2148
CVSS	0.1633	0.1669	0.1635	0.2450	0.2173	0.2173	0.2171	0.2174	0.2173	0.2174
LDRBM	0.1916	0.2042	0.2050	0.1995	0.2117	0.2719	0.2385	0.2641	0.2766	0.3045
TDRBM	**0.2058**	**0.2048**	**0.2833**	**0.2516**	**0.2728**	**0.2857**	**0.2776**	**0.2834**	**0.3460**	**0.3776**

(b) From Gabor queries to LBP search using LSH

We also compare them using the SH algorithm. To be concise, we summarize the MAP curves of using the Gabor feature to find the LBP feature with SH in Figure4.4 (c). From these results, we can draw several observations: (i) although CVSS generally performs better than CCA under LSH, it performs worst than CCA under SH, as seen in (Figure4.4 (c)). (ii) For most cases, LDRBM performs better than CCA and CVSS under LSH (Table 4.1), but the advantage is not always consistent when using small bits under SH (as shown in Figure4.4 (c)). (iii) The gap between TDRBM and LDRBM tends to increase when the number of bits increases, and the overall performance generally has a positive correlation to the number of bits used in hashing.

Color-Gabor Euclidean Search on "ImageCLEF": In this experiment, we also examine two scenarios: (i) use "Color" as query to find nearest neighbors in the "Gabor"

Table 4.2: MAP result of Color and Gabor cross feature hashing

Methods	60	70	80	90	100
CCA	0.2437	0.2388	0.2466	0.2811	0.3079
CVSS	0.1347	0.1331	0.1294	0.1304	0.1292
LDRBM	0.2601	0.2491	0.2625	0.2372	0.2516
TDRBM	**0.3282**	**0.3166**	**0.3130**	**0.3344**	**0.3298**

(a) From Color to Gabor search by SH

Methods	60	70	80	90	100
CCA	0.2011	0.2021	0.213	0.2200	0.2216
CVSS	0.1347	0.1331	0.1294	0.1304	0.1292
LDRBM	0.2118	0.2147	0.2214	0.2478	0.2780
TDRBM	**0.2473**	**0.2978**	**0.2904**	**0.3347**	**0.3068**

(b) From Color to Gabor search by LSH

Methods	60	70	80	90	100
CCA	0.2498	0.2668	0.2748	0.2931	0.3062
CVSS	0.1454	0.1415	0.1385	0.1380	0.1384
LDRBM	0.2226	0.2264	0.2412	0.2220	0.2323
TDRBM	**0.2538**	**0.2781**	**0.2934**	**0.3033**	**0.3066**

(c) From Gabor to Color search by SH

Methods	60	70	80	90	100
CCA	0.1462	0.1484	0.1534	0.1573	0.1607
CVSS	0.1454	0.1415	0.1385	0.1380	0.1384
LDRBM	0.2200	0.2383	0.2273	0.2531	0.2577
TDRBM	**0.2562**	**0.2730**	**0.3268**	**0.3010**	**0.3538**

(d) From Gabor to Color search by LSH

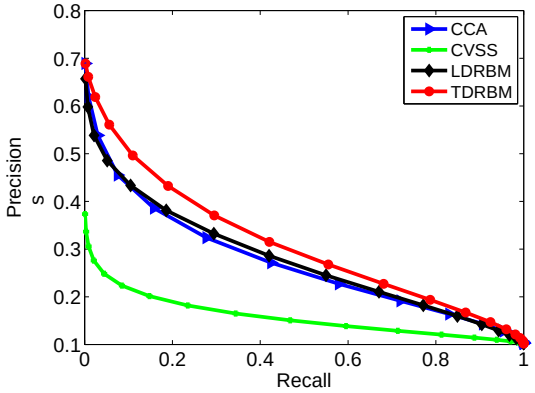
space, and (ii) vice versa. Here we choose Euclidean ℓ_2@5% as the definition of relevance ground truth. We evaluate the retrieval performance on several settings of bit sizes from 60 bits to 100 bits. The MAP results are summarized in Table 4.2. For concise, we also draw the precision and recall curves for some bits as shown in Figure 4.5. Similar results were observed as compared to the previous experiments. Besides, we also found that the performance of CVSS is not consistently better than CCA, which indicates that CVSS may not be always effective, whose empirical performance may vary across different types of data.

Discussions. From the above empirical observations, we can summarize the advantages of the proposed algorithms. First of all, both LDRBM and TDRBM in general outperform the two baselines, CCA and CVSS, for learning hash codes toward the cross-view NN search tasks. This shows that the proposed deep learning to hash scheme is able to recover in-depth nonlinear relationship between the two views as compared to the previously linear and shallow learning methods. Second, among the two models, TDRBM is empirically more accurate, stable and robust than all the algorithms including CCA, CVSS and LDRBM. This shows that it is important to explore the tight connections of the two views from the beginning of the deep training process.

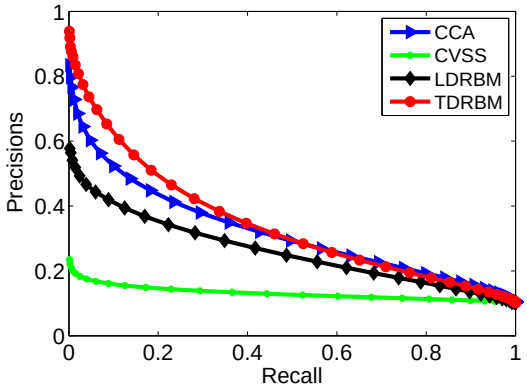
4.4.6 Experiment III: Cross-View NN Search on Multi-view Data.

In this section, we test the cross view NN search on multi-view data that consists of features from more than two views. First, we choose the "Gist" and the full set of features (excluding "Gist") generated by the "FELIB" toolbox on **ImageCLEF** dataset. Second, We use **NUS-Wide-Lite** dataset for large scale test. In **NUS-Wide-Lite** dataset, We concatenate 64-D Color histogram, 144-D Color correlogram, 225-D block-wise Color moments to form a new 433 dimension feature, we call it as the **CCC** feature. Then, we concatenate 73-D Edge direction histogram and 128-D Wavelet texture to form a 201-dimensional feature called the **EW** feature. We test cross-view NN search on both the CCC and EW features, similar to the previous experiment of cross-view NN search on two-view data. In this experiment, we also adopt semantic search (in Gist-Full test) and Euclidean search (in CCC-EW test) to fully evaluate the efficacy under different settings.

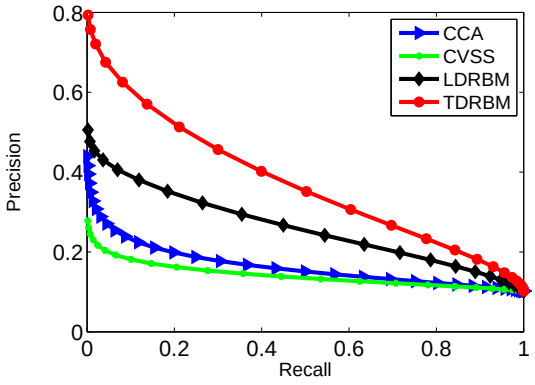
Gist-Full Semantic Search on "ImageCLEF":
We first apply the "Gist" feature to retrieve instances from the full feature space by "FELIB", and then do another evaluation by reversing the order. Table 4.3 shows the MAP results. We can see that CCA always is the worst among all. When using a small number of bits, CVSS also exceeds LDRBM. When the number of bits increases,



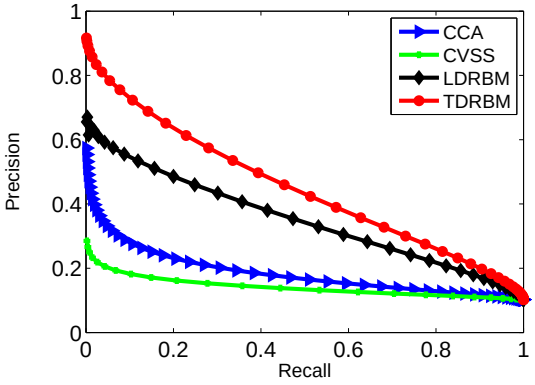
4.5.a: search by SH with 40 bits



4.5.b: search by SH with 100 bits



4.5.c: search by LSH with 100 bits



4.5.d: search by LSH with 200 bits

Figure 4.5: Using LSH cross hashing Color and Gabor feature on ImageCLEF dataset. (a) and (b) are from Color to Gabor. (c) and (d) are from Gabor to Color

CVSS performs slightly worse. For the deep learning to hash algorithms, both LDRBM and TDRBM generally perform better when the number of bits increases, especially for TDRBM. When using a large bit size to perform hashing (as shown in the lat column of Table 4.3), TDRBM is significantly better than the other three methods.

CCC-EW Euclidean Search on "NUS-Wide-Lite":
We first apply the color-related feature "CCC" to search for texture-related feature "EW", and then reverse the order. We adopt ℓ_2@10% Euclidean distance for the relevance definition. In the training phase, we train the model using the entire training set (consisting of 27807 images) given in **NUS-Wide-Lite** dataset. In the test phase, we randomly choose $10,000$ queries on the test set to perform SH and LSH respectively. We test on both small bit sizes (from 8-bit to 48-bit) and large bit sizes (from 80-bit to 200-bit). For each experiment, once the query set is selected, we run the program 3 times and report the average results. For space limitation, we report one hashing method for each cross-view NN search task (i.e., use CCC to retrieve EW under SH and use EW to retrieve CCC under LSH) to cover two diverse settings of the whole experiment. Table 4.4 shows the evaluation details of MAP results. Similar to the previous experiments, our TDRBM model always achieves the best performances among most of different settings. This experiment further validates that the proposed TDRBM model is more effective and far more robust than the other baselines under different datasets of varied feature spaces.

Table 4.3: MAP performance of cross-view hashing between GIST and full feature set.

Methods	40	60	80	100	200
CCA	0.1648	0.1663	0.1678	0.1695	0.1749
CVSS	0.2523	0.2423	0.2361	0.2312	0.2172
LDRBM	0.2419	0.2538	0.2402	0.2699	0.3517
TDRBM	**0.2712**	**0.2995**	**0.3263**	**0.3690**	**0.4299**

(a) From GIST to full feature space

Methods	40	60	80	100	200
CCA	0.1634	0.1647	0.1657	0.1664	0.1711
CVSS	0.2525	0.2424	0.2361	0.2312	0.2173
LDRBM	0.2181	0.2589	0.2672	0.2786	0.3196
TDRBM	**0.2882**	**0.3145**	**0.3493**	**0.3501**	**0.4300**

(b) From full feature space to GIST

Table 4.4: MAP performance of cross-view hashing between CCC and EW.

Methods	8	16	24	48	80	100	200
CCA	0.133	0.129	0.116	0.123	0.133	0.134	0.135
CVSS	0.132	0.129	0.125	0.118	0.115	0.114	0.110
LDRBM	0.128	0.143	0.155	0.154	0.162	0.165	0.155
TDRBM	**0.157**	**0.164**	**0.168**	**0.178**	**0.191**	**0.190**	**0.204**

(a) From CCC to EW search using SH

Methods	8	16	24	48	80	100	200
CCA	0.101	0.102	0.1019	0.103	0.103	0.104	0.105
CVSS	0.114	0.114	0.113	0.110	0.108	0.107	0.105
LDRBM	0.110	0.123	0.131	0.131	0.147	0.1549	0.174
TDRBM	**0.120**	**0.128**	**0.143**	**0.167**	**0.175**	**0.197**	**0.215**

(b) From EW to CCC search using LSH

4.5 Conclusions

This work investigated a new deep learning to hash framework for cross-view nearest neighbor search towards multimedia search and mining applications. In particular, this work comprehensively explored two different deep RBM models for cross-view learning to hash: loose deep RBMs and tight deep RBMs models. Through learning the shared representations on data instances with different views, empirical results showed that the image retrieval performance of the subsequent learning to hash task can be significantly boosted in comparison to the state-of-the-art approaches for cross-view learning to hash with the conventional shallow learning scheme, and the TDRBM model is generally more effective than the LDRBM model. For future work, although this work assumes cross-view learning by dealing with only two input views and two-layer model structure, the proposed technique can be easily extended to handle multi-view input data and multi-layer model structure for cross-view/multi-view data mining tasks in a variety of real-world scenarios.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, I mainly introduce the survey of learning to hash and deep learning. I introduce some algorithms, fundamental theories, and also some experiment results. I also introduce our own work of cross view hashing using deep learning. There are a lot other work that can be done for improvement. I introduce them in the next section.

5.2 Future Work

Deep learning has a variety of applications in a wide machine learning fields. Here are some other ideas I get when I do my research.

5.2.1 Image and Text Cross View Hashing

The future work can focus on the cross view image-text hashing(given image, return text. given text, return image), which is more related to the reality. Since the current multi-view feature work has some drawbacks. 1, it is not quite common that we need to use one feature to find another feature. So in image and text retrieval, we can treat image as one 'feature' and text as another 'feature', since both image and text describe the same thing. 2, My original design of the experiment is not just test deep model's ability on cross-view image feature retrieval, instead, I want to test its ability that holds relationships between different properties of one item. such as its ability that given one attribute of a item, find other attributes of the same item. In this experiment, I just choose different image features as different attributes of the same item (image) and it works well. So, this work can be extended to image-text hashing.

The work I did is different but very related to two existing works. [TFW08] and [SS12]. [TFW08] uses multi-layer autoencoder to construct single-view hashing algorithm. Our

work is a multi-view hashing algorithm. [SS12] trys to use text to infer images and use images to infer text, but they doesn't benefit from learning to hash technology. For a large dataset, such as more than one million, if they want to infer images using text, they must use text to generate a pseudo image, when compare this pseudo image with the other 1 million images and return the nearest neighbors, which is infeasible in large dataset. While, our method is, project all text and images into the binary hamming space and given one query, can project the query in the same hamming space and retrieval. Of course, can use some accelerating technique such as multi-index hashing [NPF12] in Hamming space. I believe this method will be more practical and should work fine.

5.2.2 Heterogeneous Transfer Learning

Deep learning can also be used for transfer learning [MD10, DB11, CXWS12, MDG+12], as mentioned in chapter 2. I think some work of domain adaptation can also benefit from deep transfer learning. Such as [DXT12]. In [DXT12], the authors use SVM to do heterogeneous domain adaptation. They transfer text from one domain to another text domain to help text classification. They transfer images from image domain to another image domain to help image classification. But using SVM, in the current framework, they may not be able to do homogeneous domain adaptation. That means they may not be able to transfer text to image domain and help image classification or transfer image to text domain to help text classification.

But, using our framework, it is totally feasible to do homogeneous domain adaptation. Our framework is like this(This is a joint idea with Dr.Peilin Zhao):

Assume $\phi(\mathbf{x};\theta)$ is a net

Objective

$$\min \mathbf{w}, b, \theta, \mathbf{x}i_{s,i}, \mathbf{x}i_{t,i} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C(\sum_{i=1}^{n_s} \mathbf{x}i_{s,i} + \sum_{i=1}^{n_t} \mathbf{x}i_{t,i}) + \Omega(\theta) \tag{5.1}$$

$$\text{s.t.} \quad y_{s,i}(\mathbf{w}^\top \phi(\mathbf{x}_{s,i};\theta) + b) \geq 1 - \mathbf{x}i_{s,i}, \mathbf{x}i_{s,i} \geq 0 \tag{5.2}$$

$$y_{t,i}(\mathbf{w}^\top \phi(\mathbf{x}_{t,i};\theta) + b) \geq 1 - \mathbf{x}i_{t,i}, \mathbf{x}i_{t,i} \geq 0 \tag{5.3}$$

Fist choice

$$\Omega(\theta) = \|\theta\|^2 \tag{5.4}$$

Second choice

$$\Omega(\theta) = KL(\mathcal{N}(0;\Sigma)|\mathcal{N}(0;K(\theta))) \tag{5.5}$$

where Σ is prior information on instances and $K(\theta) = [\phi(x_i;\theta)^\top \phi(x_j;\theta)]$.

Also, $\Omega(\theta)$ can be other choices.

Protocol 1 Heterogeneous Transfer Learning

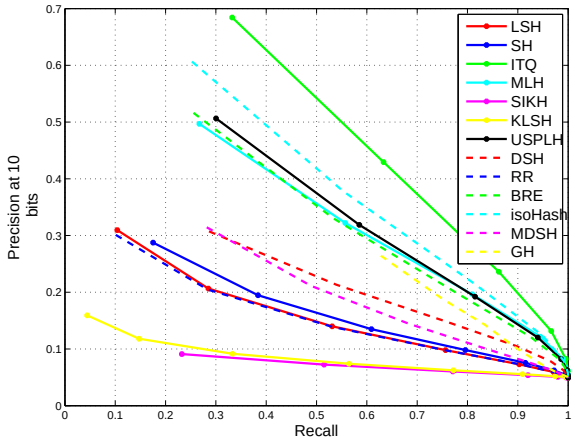
Initialize $\mathbf{w}$, b, and θ
for t=1...T **do**
  Fix θ to learn $\mathbf{w}, b$
  Fix $\mathbf{w}, b$ to learn θ
end for

Appendix A

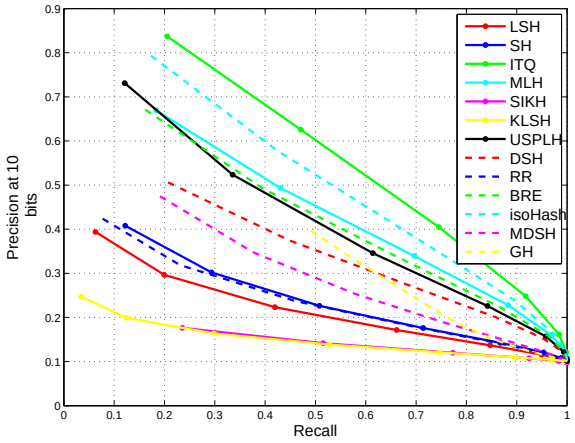
Appendix

A.1 Results of learning to hash algorithms on other dataset

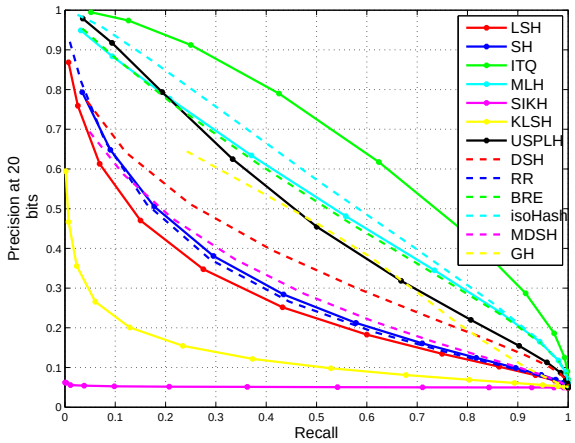
A.1.1 MNIST5k and Notredame5k dataset



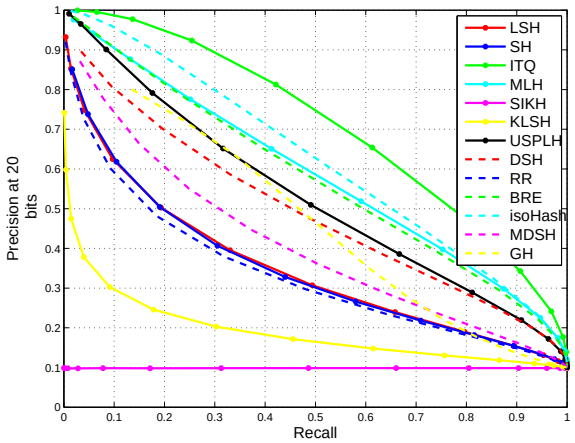
A.1.a: 10 bits



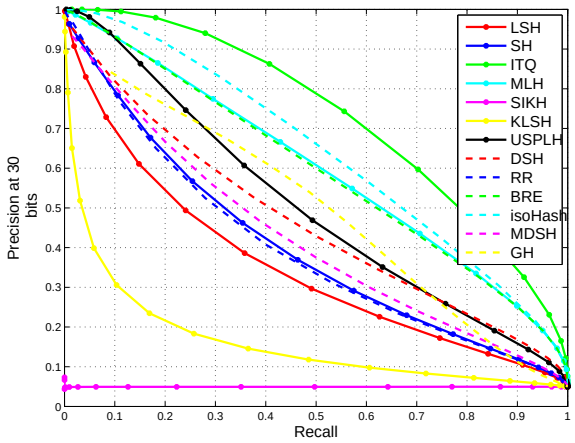
A.1.b: 10 bits



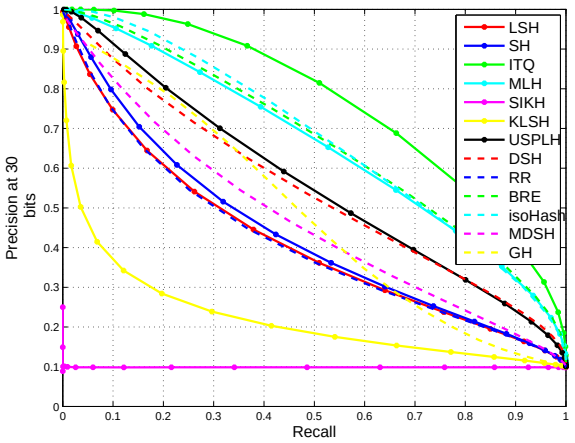
A.1.c: 20 bits



A.1.d: 20 bits

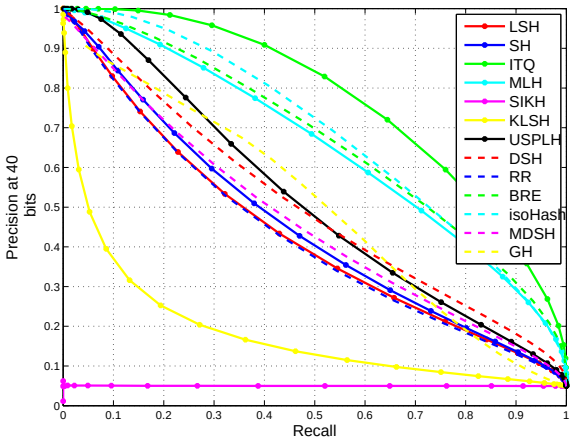


A.1.e: 30 bits

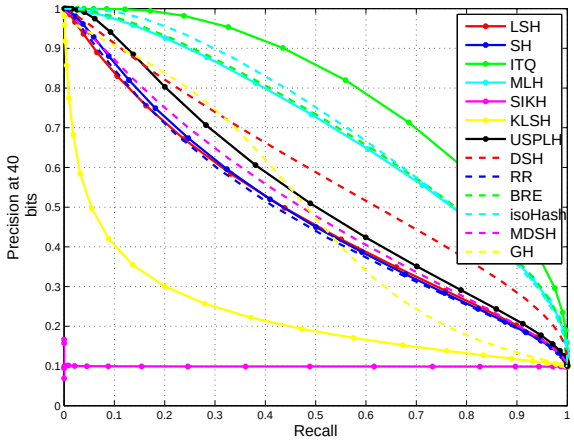


A.1.f: 30 bits

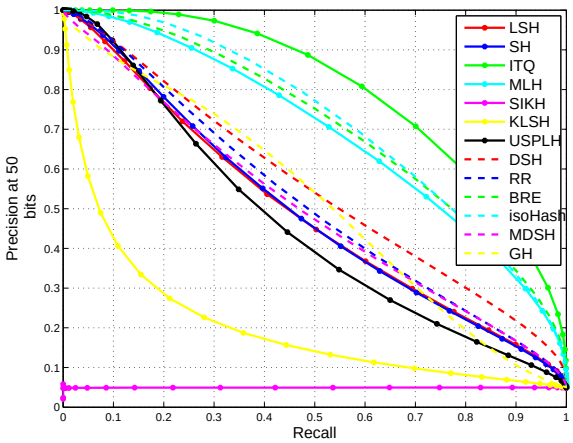
Figure A.1: MNIST5k Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10% .



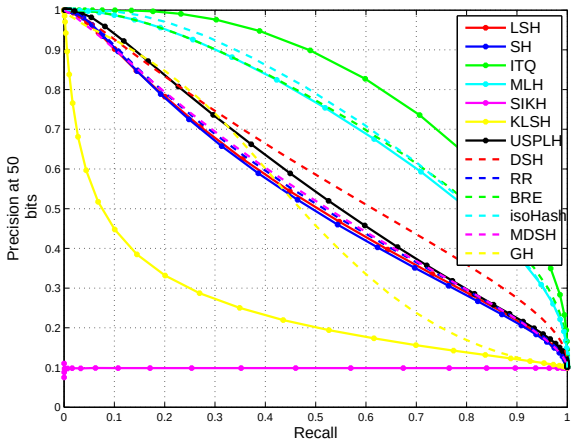
A.2.a: 40 bits



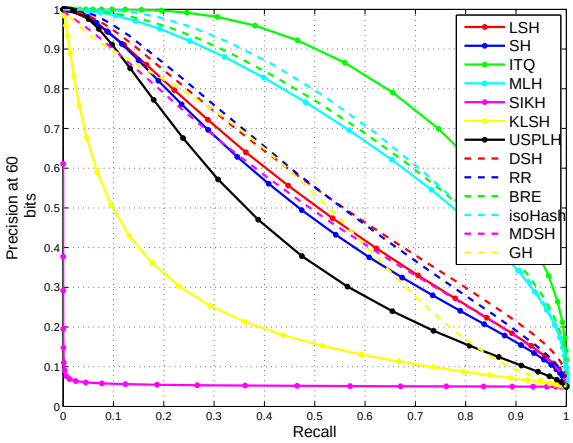
A.2.b: 40 bits



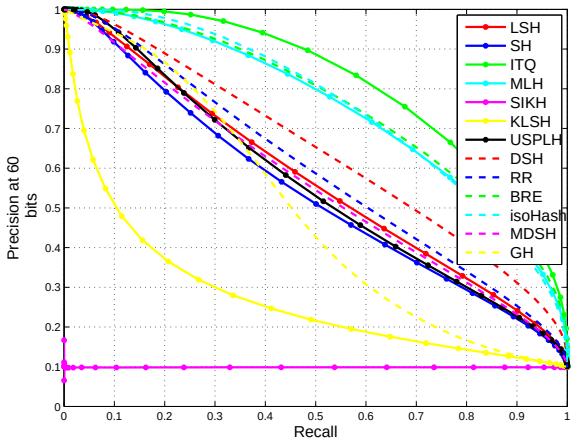
A.2.c: 50 bits



A.2.d: 50 bits

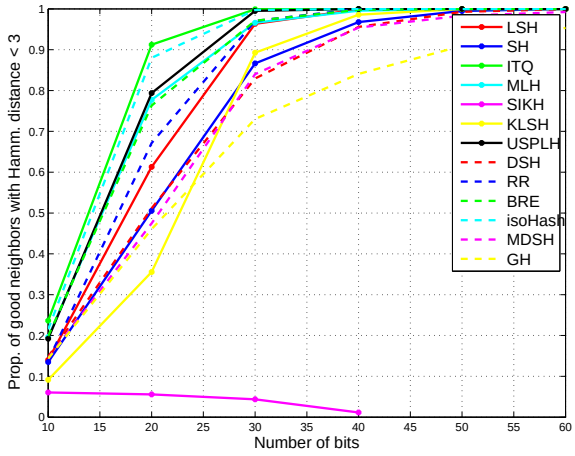


A.2.e: 60 bits

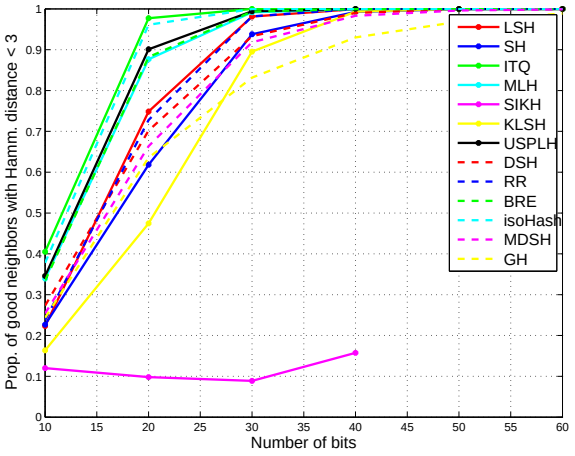


A.2.f: 60 bits

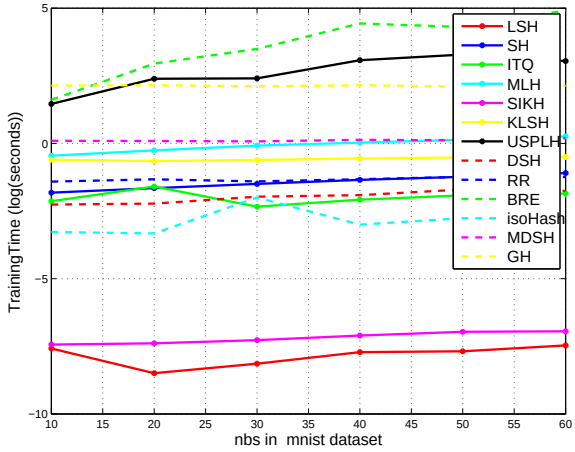
Figure A.2: MNIST5k Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10% .



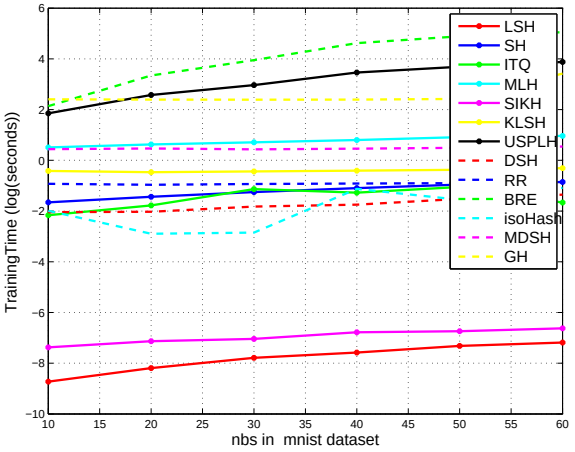
A.3.a: Precision in Hamming radius 3



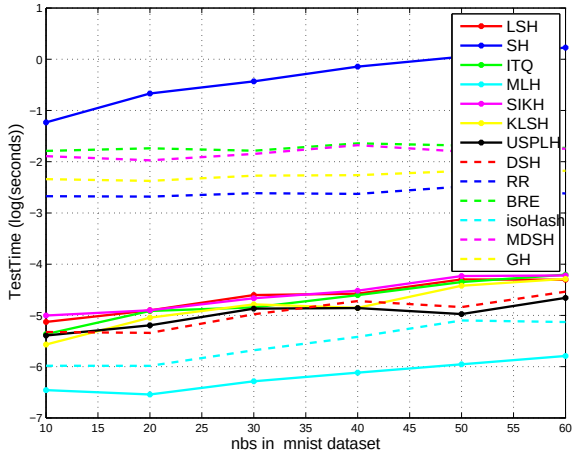
A.3.b: Precision in Hamming radius 3



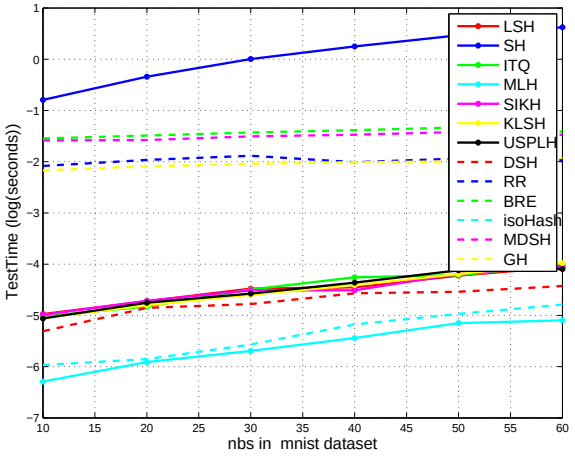
A.3.c: Training Time (log)



A.3.d: Training Time (log)



A.3.e: Compress Time (log)



A.3.f: Compress Time (log)

Figure A.3: MNIST5k Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10%.

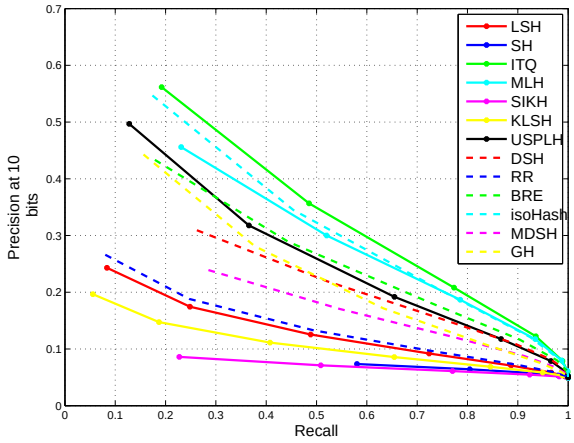
Table A.1: MAP in MNIST5k dataset

Methods	10	20	30	40	50	60
LSH	0.11 ± 0.01	0.22 ± 0.01	0.29 ± 0.00	0.33 ± 0.00	0.36 ± 0.00	0.37 ± 0.00
SH	0.11 ± 0.00	0.19 ± 0.01	0.25 ± 0.00	0.28 ± 0.01	0.31 ± 0.01	0.32 ± 0.01
ITQ	**0.18 ± 0.00**	**0.30 ± 0.00**	**0.34 ± 0.00**	**0.37 ± 0.00**	**0.38 ± 0.00**	**0.39 ± 0.00**
MLH	0.15 ± 0.00	0.26 ± 0.00	0.30 ± 0.01	0.33 ± 0.01	0.35 ± 0.00	0.36 ± 0.00
SIKH	0.06 ± 0.00	0.05 ± 0.00	0.05 ± 0.01	0.05 ± 0.00	0.05 ± 0.00	0.09 ± 0.01
KLSH	0.08 ± 0.00	0.15 ± 0.01	0.24 ± 0.01	0.28 ± 0.01	0.31 ± 0.01	0.32 ± 0.02
USPLH	0.15 ± 0.01	0.26 ± 0.01	0.32 ± 0.01	0.34 ± 0.01	**0.38 ± 0.00**	**0.39 ± 0.00**
DSH	0.11 ± 0.00	0.19 ± 0.01	0.24 ± 0.01	0.27 ± 0.01	0.30 ± 0.00	0.31 ± 0.00
RR	0.11 ± 0.00	0.23 ± 0.00	0.30 ± 0.01	0.33 ± 0.00	0.36 ± 0.00	0.38 ± 0.00
BRE	0.15 ± 0.00	0.26 ± 0.00	0.30 ± 0.00	0.33 ± 0.00	0.35 ± 0.00	0.36 ± 0.00
isoHash	0.17 ± 0.00	0.28 ± 0.00	0.34 ± 0.00	0.36 ± 0.00	**0.38 ± 0.00**	**0.39 ± 0.00**
MDSH	0.11 ± 0.00	0.18 ± 0.00	0.25 ± 0.01	0.29 ± 0.00	0.31 ± 0.00	0.32 ± 0.00
GH	0.10 ± 0.00	0.18 ± 0.01	0.23 ± 0.00	0.26 ± 0.00	0.28 ± 0.00	0.31 ± 0.00

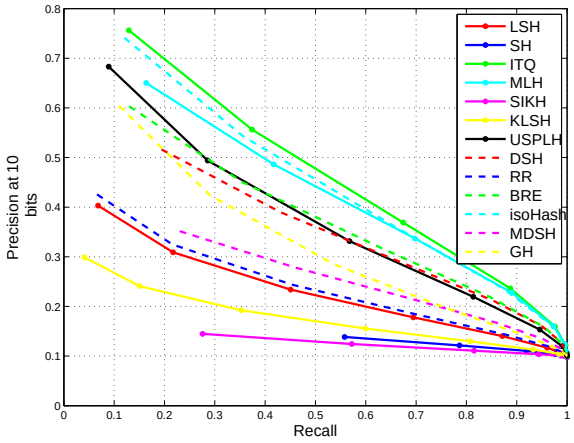
(a) MAP result of mnist5k Dataset at ℓ_2@5%

Methods	10	20	30	40	50	60
LSH	0.17 ± 0.01	0.30 ± 0.00	0.36 ± 0.00	0.40 ± 0.00	0.42 ± 0.00	0.44 ± 0.00
SH	0.18 ± 0.01	0.27 ± 0.01	0.33 ± 0.00	0.36 ± 0.00	0.38 ± 0.00	0.39 ± 0.00
ITQ	**0.28 ± 0.00**	**0.38 ± 0.00**	**0.42 ± 0.00**	**0.44 ± 0.00**	**0.45 ± 0.00**	**0.46 ± 0.00**
MLH	0.24 ± 0.00	0.35 ± 0.00	0.39 ± 0.00	0.41 ± 0.00	0.43 ± 0.00	0.44 ± 0.00
SIKH	0.11 ± 0.01	0.10 ± 0.00	0.11 ± 0.00	0.10 ± 0.01	0.10 ± 0.01	0.10 ± 0.01
KLSH	0.14 ± 0.00	0.22 ± 0.01	0.30 ± 0.00	0.33 ± 0.01	0.35 ± 0.01	0.38 ± 0.00
USPLH	0.25 ± 0.00	0.35 ± 0.01	0.38 ± 0.01	0.41 ± 0.00	0.38 ± 0.01	0.41 ± 0.00
DSH	0.20 ± 0.01	0.29 ± 0.01	0.34 ± 0.01	0.37 ± 0.01	0.38 ± 0.01	0.40 ± 0.01
RR	0.18 ± 0.00	0.29 ± 0.01	0.36 ± 0.01	0.40 ± 0.00	0.42 ± 0.00	0.44 ± 0.00
BRE	0.24 ± 0.01	0.35 ± 0.00	0.39 ± 0.00	0.41 ± 0.00	0.43 ± 0.00	0.43 ± 0.00
isoHash	0.27 ± 0.00	0.37 ± 0.00	0.41 ± 0.00	**0.44 ± 0.00**	**0.45 ± 0.00**	**0.46 ± 0.00**
MDSH	0.19 ± 0.00	0.28 ± 0.00	0.34 ± 0.00	0.37 ± 0.00	0.39 ± 0.00	0.40 ± 0.00
GH	0.18 ± 0.00	0.27 ± 0.01	0.31 ± 0.01	0.35 ± 0.01	0.37 ± 0.00	0.39 ± 0.00

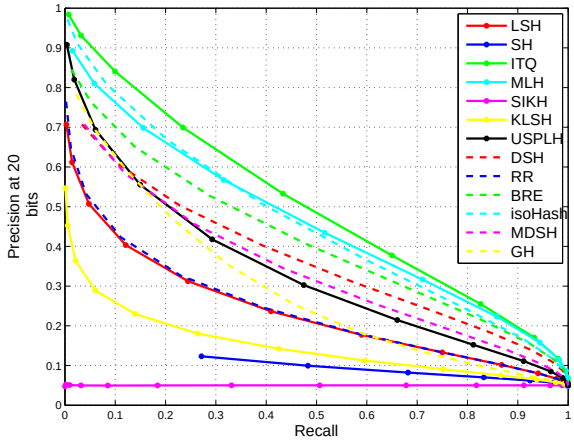
(b) MAP result of MNIST5k Dataset at ℓ_2@10%



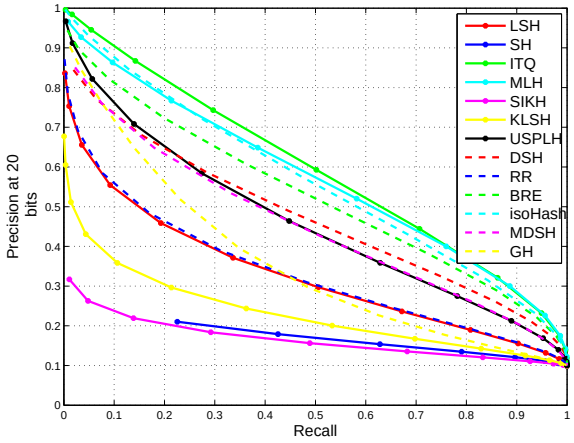
A.4.a: 10 bits



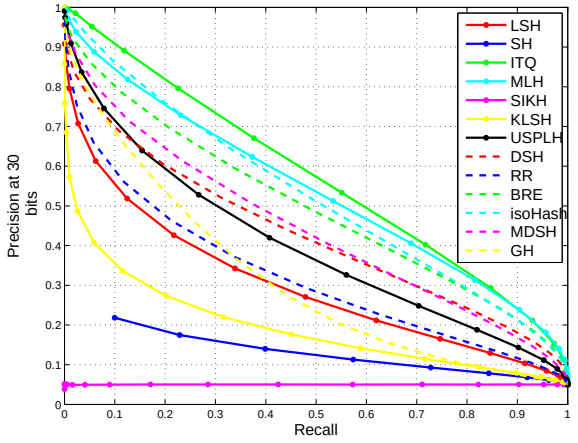
A.4.b: 10 bits



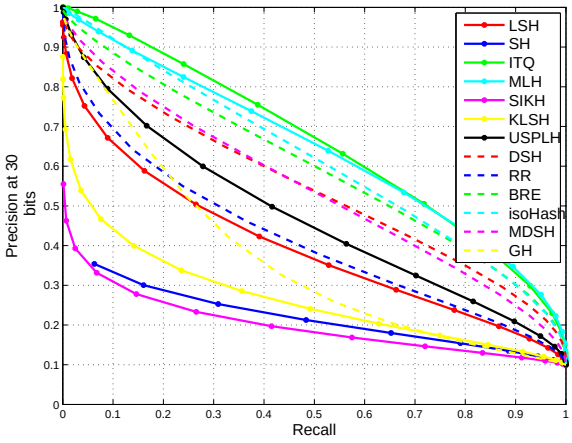
A.4.c: 20 bits



A.4.d: 20 bits

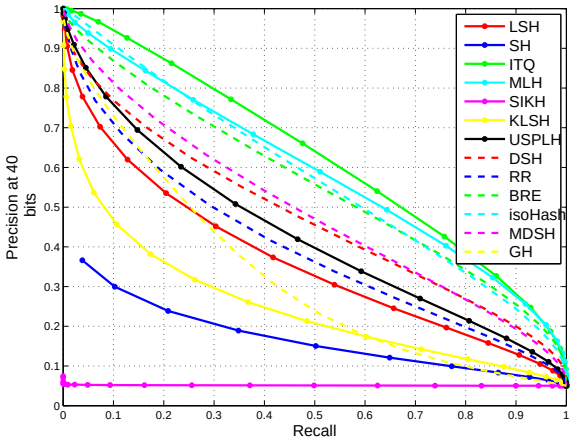


A.4.e: 30 bits

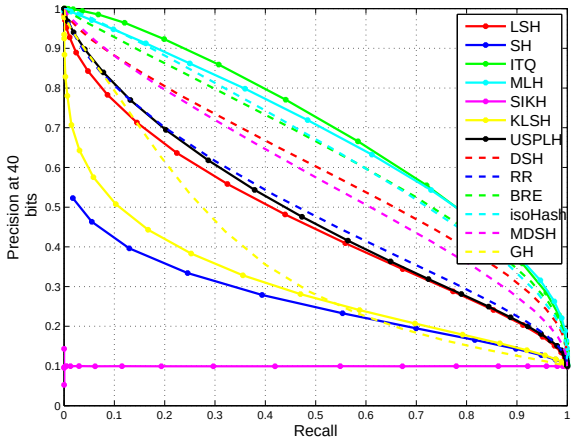


A.4.f: 30 bits

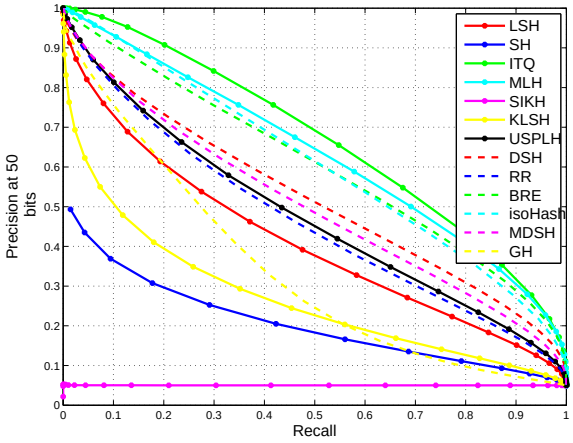
Figure A.4: Notredame5k Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10% .



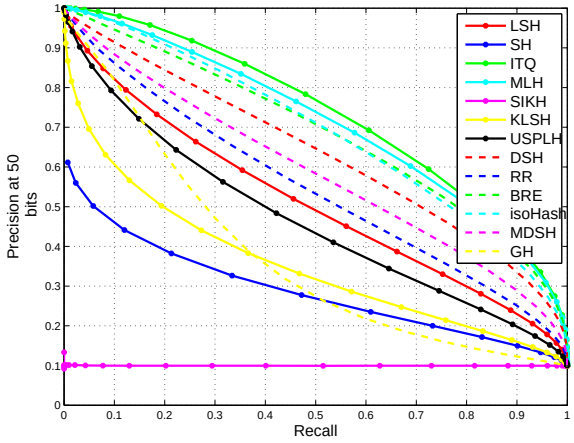
A.5.a: 40 bits



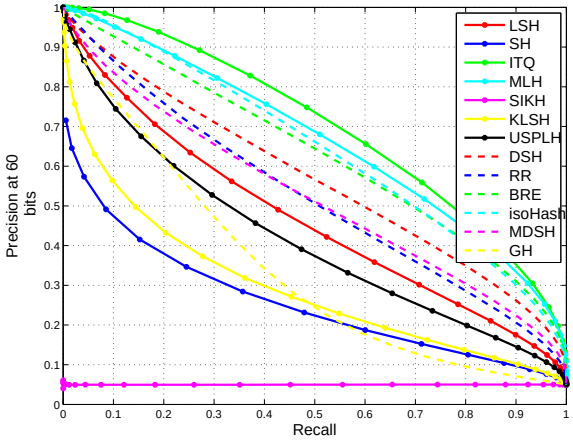
A.5.b: 40 bits



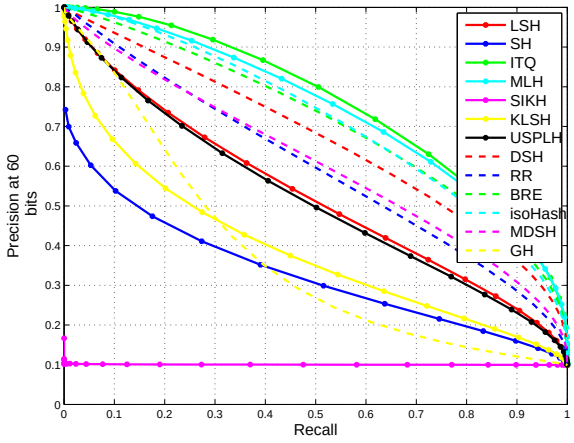
A.5.c: 50 bits



A.5.d: 50 bits

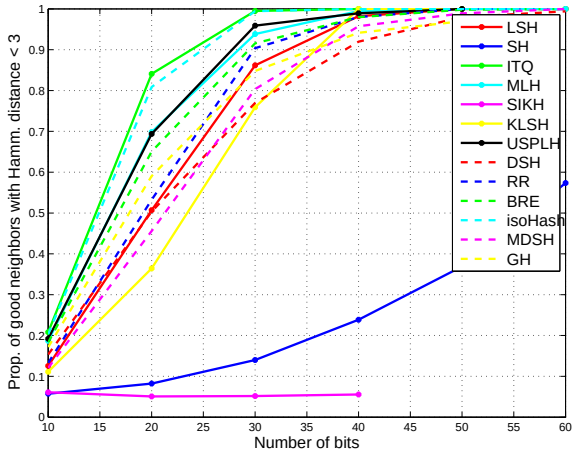


A.5.e: 60 bits

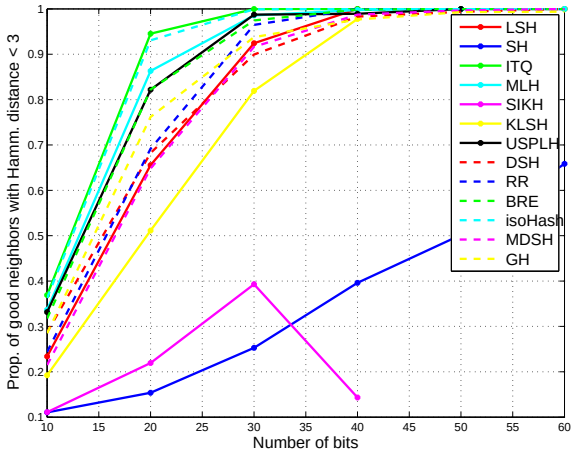


A.5.f: 60 bits

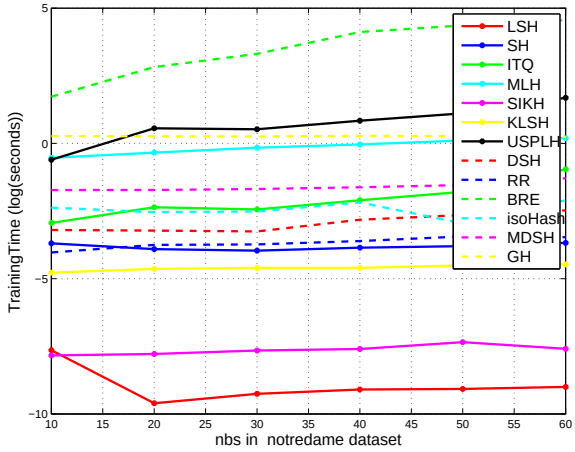
Figure A.5: Notredame5k Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10% .



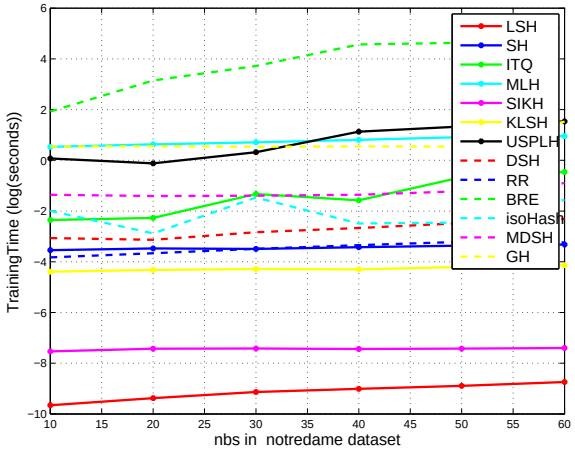
A.6.a: Precision in Hamming radius 3



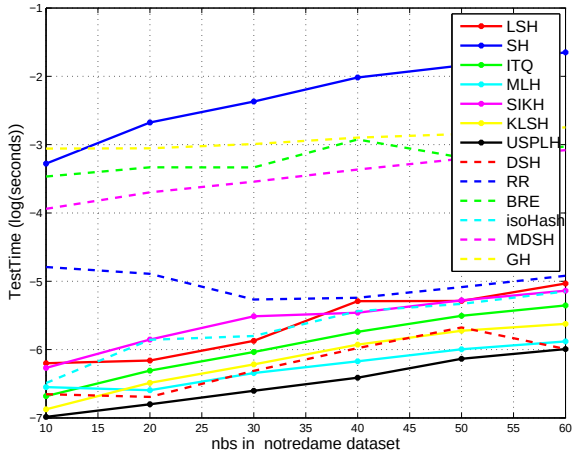
A.6.b: Precision in Hamming radius 3



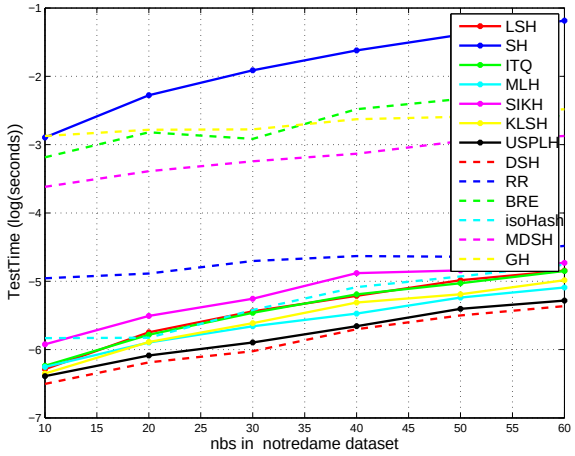
A.6.c: Training Time (log)



A.6.d: Training Time (log)



A.6.e: Compress Time (log)



A.6.f: Compress Time (log)

Figure A.6: Notredame5k Data Precision and Recall Curve, left column is ℓ_2@5%, right column is ℓ_2@10%.

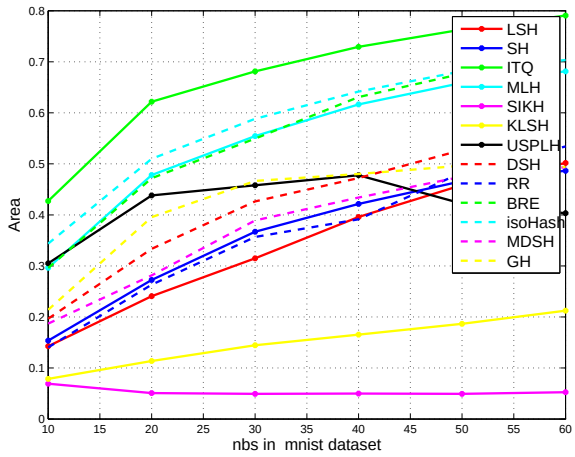
Table A.2: MAP in Notredame5k dataset

Methods	10	20	30	40	50	60
LSH	0.10 ± 0.00	0.19 ± 0.01	0.26 ± 0.00	0.31 ± 0.00	0.33 ± 0.00	0.34 ± 0.01
SH	0.06 ± 0.00	0.06 ± 0.00	0.07 ± 0.00	0.08 ± 0.00	0.09 ± 0.01	0.11 ± 0.01
ITQ	**0.16 ± 0.00**	**0.28 ± 0.00**	**0.33 ± 0.00**	**0.36 ± 0.00**	**0.37 ± 0.00**	**0.38 ± 0.01**
MLH	0.14 ± 0.00	0.24 ± 0.01	0.29 ± 0.00	0.32 ± 0.01	0.35 ± 0.01	0.36 ± 0.01
SIKH	0.06 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00
KLSH	0.09 ± 0.00	0.15 ± 0.01	0.23 ± 0.00	0.28 ± 0.01	0.31 ± 0.02	0.32 ± 0.01
USPLH	0.15 ± 0.00	0.24 ± 0.02	0.30 ± 0.01	0.33 ± 0.01	0.34 ± 0.01	0.34 ± 0.02
DSH	0.11 ± 0.01	0.19 ± 0.00	0.24 ± 0.01	0.27 ± 0.01	0.30 ± 0.01	0.32 ± 0.01
RR	0.10 ± 0.00	0.20 ± 0.00	0.28 ± 0.00	0.32 ± 0.00	0.34 ± 0.01	0.35 ± 0.01
BRE	0.14 ± 0.00	0.23 ± 0.01	0.28 ± 0.00	0.31 ± 0.00	0.33 ± 0.00	0.35 ± 0.00
isoHash	**0.16 ± 0.00**	0.27 ± 0.00	0.32 ± 0.00	0.35 ± 0.00	0.36 ± 0.01	**0.38 ± 0.00**
MDSH	0.09 ± 0.00	0.17 ± 0.00	0.23 ± 0.00	0.26 ± 0.00	0.28 ± 0.00	0.29 ± 0.00
GH	0.13 ± 0.00	0.21 ± 0.00	0.26 ± 0.00	0.30 ± 0.00	0.32 ± 0.00	0.34 ± 0.00

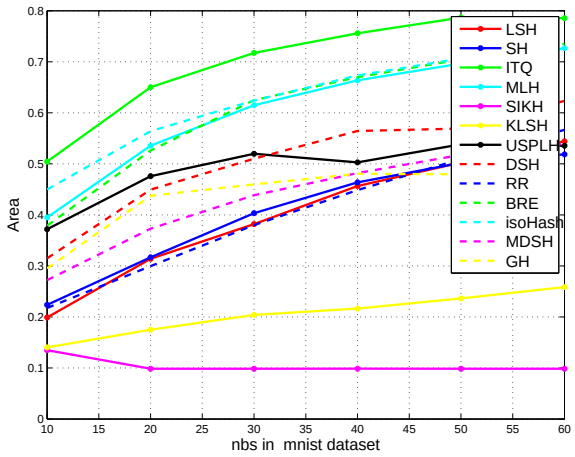
(a) MAP result of Notredame5k Dataset at ℓ_2@5%

Methods	10	20	30	40	50	60
LSH	0.18 ± 0.00	0.28 ± 0.00	0.34 ± 0.01	0.39 ± 0.01	0.41 ± 0.01	0.42 ± 0.01
SH	0.11 ± 0.00	0.12 ± 0.00	0.13 ± 0.00	0.15 ± 0.00	0.16 ± 0.00	0.17 ± 0.01
ITQ	**0.26 ± 0.00**	**0.37 ± 0.00**	**0.41 ± 0.00**	**0.43 ± 0.00**	**0.45 ± 0.00**	**0.45 ± 0.01**
MLH	0.24 ± 0.00	0.34 ± 0.00	0.39 ± 0.00	0.41 ± 0.00	0.43 ± 0.00	0.44 ± 0.00
SIKH	0.11 ± 0.00	0.14 ± 0.01	0.16 ± 0.01	0.10 ± 0.00	0.11 ± 0.01	0.10 ± 0.01
KLSH	0.15 ± 0.01	0.23 ± 0.00	0.30 ± 0.02	0.33 ± 0.01	0.37 ± 0.01	0.39 ± 0.02
USPLH	0.24 ± 0.00	0.33 ± 0.00	0.38 ± 0.00	0.36 ± 0.01	0.41 ± 0.00	0.41 ± 0.02
DSH	0.21 ± 0.00	0.29 ± 0.01	0.34 ± 0.01	0.38 ± 0.00	0.40 ± 0.00	0.41 ± 0.00
RR	0.19 ± 0.00	0.29 ± 0.01	0.36 ± 0.01	0.40 ± 0.00	0.41 ± 0.02	0.43 ± 0.01
BRE	0.23 ± 0.00	0.33 ± 0.00	0.38 ± 0.00	0.40 ± 0.00	0.42 ± 0.00	0.43 ± 0.00
isoHash	**0.26 ± 0.00**	0.36 ± 0.00	**0.41 ± 0.00**	**0.43 ± 0.00**	**0.45 ± 0.00**	**0.45 ± 0.01**
MDSH	0.17 ± 0.00	0.27 ± 0.00	0.32 ± 0.00	0.35 ± 0.00	0.36 ± 0.00	0.37 ± 0.00
GH	0.22 ± 0.00	0.30 ± 0.00	0.35 ± 0.00	0.37 ± 0.00	0.39 ± 0.00	0.41 ± 0.00

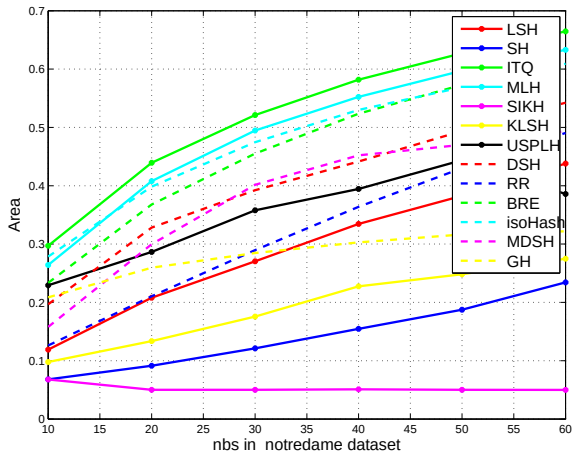
(b) MAP result of Notredame5k Dataset at ℓ_2@10%



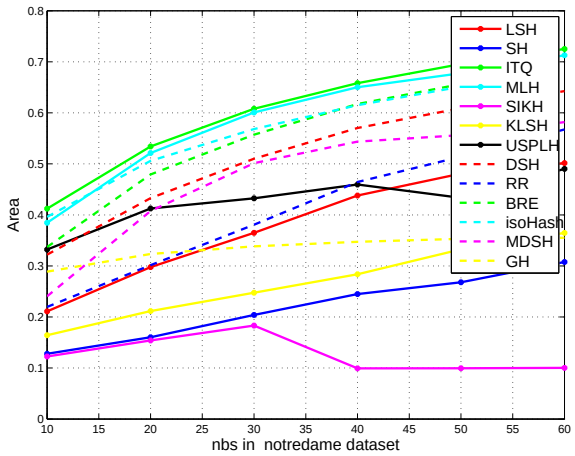
A.7.a: mnist5k



A.7.b: mnist5k



A.7.c: notredame5k



A.7.d: notredame5k

Figure A.7: Area Under P-R Curve, left column is ℓ_2@5%, right column is ℓ_2@10% .

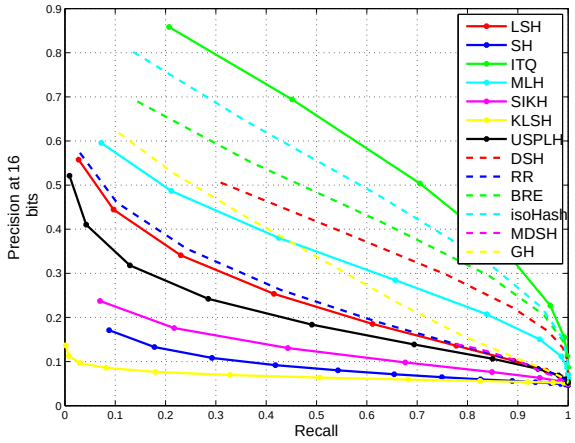
Table A.3: MAP result of CIFAR10 Dataset at ℓ_2@10%

Methods	16	24	32	64	128
LSH	0.16 ± 0.02	0.21 ± 0.01	0.27 ± 0.01	0.34 ± 0.00	**0.36 ± 0.01**
SH	0.07 ± 0.01	0.08 ± 0.01	0.09 ± 0.02	0.12 ± 0.03	0.16 ± 0.03
ITQ	**0.21 ± 0.00**	**0.26 ± 0.01**	**0.29 ± 0.00**	**0.34 ± 0.00**	**0.36 ± 0.01**
MLH	0.17 ± 0.01	0.21 ± 0.01	0.24 ± 0.00	0.30 ± 0.00	0.32 ± 0.01
SIKH	0.08 ± 0.01	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00
KLSH	0.07 ± 0.00	0.06 ± 0.00	0.05 ± 0.00	0.06 ± 0.01	0.06 ± 0.01
USPLH	0.15 ± 0.01	0.19 ± 0.01	0.21 ± 0.00	0.31 ± 0.02	0.33 ± 0.05
DSH	0.15 ± 0.00	0.18 ± 0.00	0.21 ± 0.00	0.25 ± 0.01	0.28 ± 0.01
RR	0.16 ± 0.03	0.23 ± 0.01	0.27 ± 0.01	**0.34 ± 0.01**	0.35 ± 0.01
BRE	0.18 ± 0.00	0.23 ± 0.01	0.26 ± 0.00	0.31 ± 0.00	0.33 ± 0.00
isoHash	0.20 ± 0.01	**0.26 ± 0.01**	**0.29 ± 0.00**	**0.34 ± 0.00**	0.35 ± 0.00
MDSH	0.09 ± 0.00	0.09 ± 0.00	0.09 ± 0.00	0.10 ± 0.00	0.11 ± 0.00
GH	0.18 ± 0.01	0.23 ± 0.00	0.26 ± 0.01	0.31 ± 0.01	0.34 ± 0.01

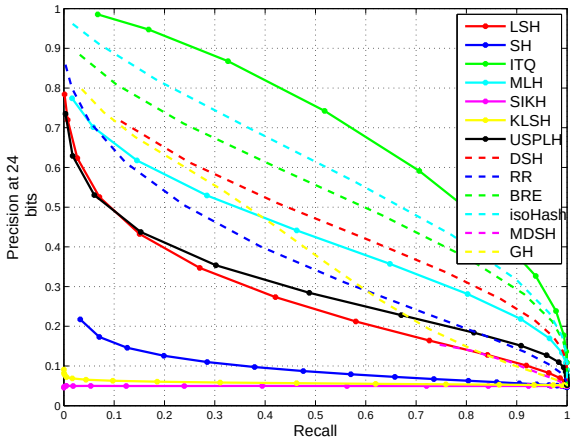
Table A.4: MAP result of Caltech101 Dataset at ℓ_2@10%

Methods	16	24	32	64	128
LSH	0.18 ± 0.01	0.24 ± 0.00	0.29 ± 0.00	0.36 ± 0.00	0.39 ± 0.01
SH	0.05 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	0.07 ± 0.00
ITQ	**0.24 ± 0.00**	**0.29 ± 0.00**	**0.32 ± 0.00**	**0.37 ± 0.00**	0.39 ± 0.00
MLH	0.21 ± 0.00	0.25 ± 0.01	0.28 ± 0.00	0.34 ± 0.00	0.37 ± 0.00
SIKH	0.06 ± 0.01	0.05 ± 0.00	0.05 ± 0.01	0.15 ± 0.01	0.28 ± 0.02
KLSH	0.08 ± 0.01	0.09 ± 0.01	0.15 ± 0.01	0.28 ± 0.00	0.35 ± 0.01
USPLH	0.18 ± 0.02	0.24 ± 0.01	0.29 ± 0.00	0.29 ± 0.02	0.20 ± 0.01
DSH	0.17 ± 0.00	0.20 ± 0.00	0.22 ± 0.01	0.26 ± 0.01	0.29 ± 0.01
RR	0.17 ± 0.01	0.25 ± 0.01	0.29 ± 0.00	**0.37 ± 0.01**	**0.40 ± 0.00**
BRE	0.20 ± 0.00	0.25 ± 0.01	0.28 ± 0.01	0.33 ± 0.00	0.35 ± 0.00
isoHash	0.22 ± 0.01	0.28 ± 0.01	0.31 ± 0.00	0.36 ± 0.00	0.39 ± 0.00
MDSH	0.13 ± 0.00	0.17 ± 0.01	0.19 ± 0.01	0.30 ± 0.00	0.35 ± 0.00
GH	0.16 ± 0.01	0.20 ± 0.00	0.24 ± 0.02	0.32 ± 0.00	0.35 ± 0.01

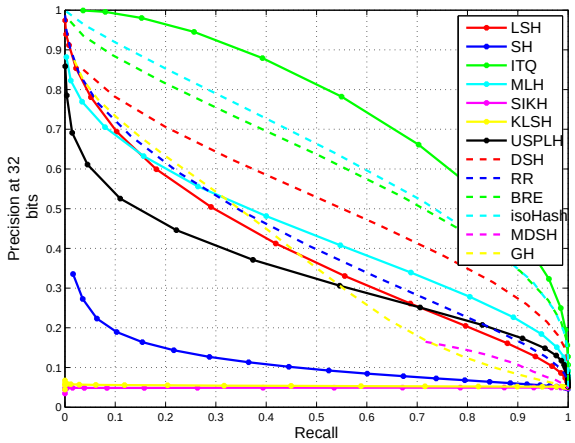
A.1.2 CIFAR10 and Caltech101 dataset



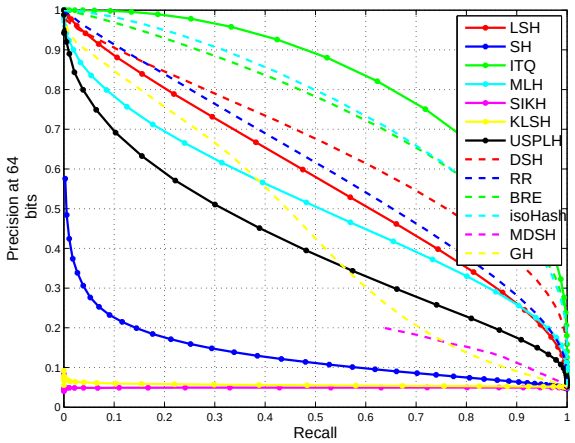
A.8.a: 16 bits



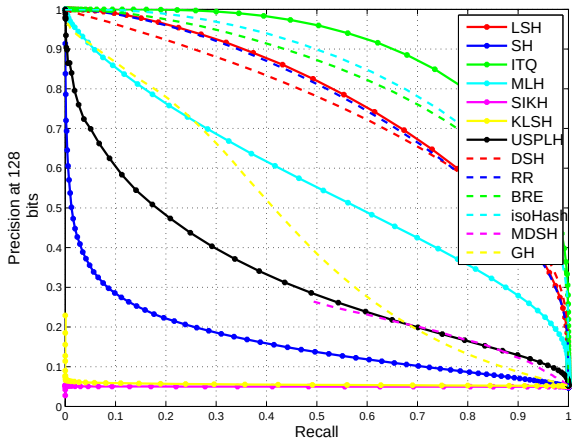
A.8.b: 24 bits



A.8.c: 32 bits

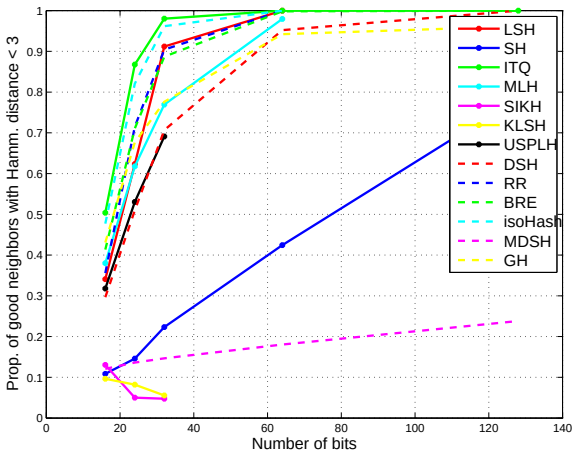


A.8.d: 64 bits

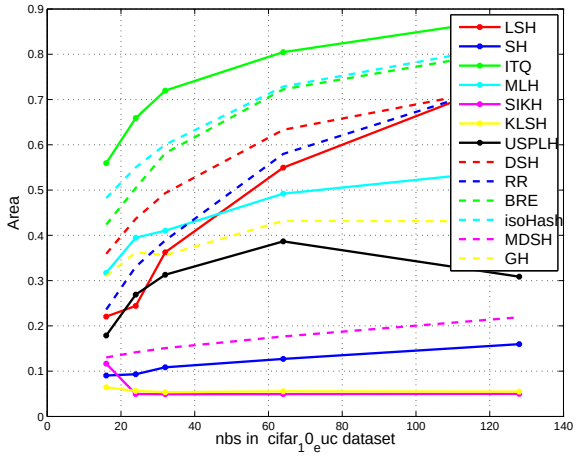


A.8.e: 128 bits

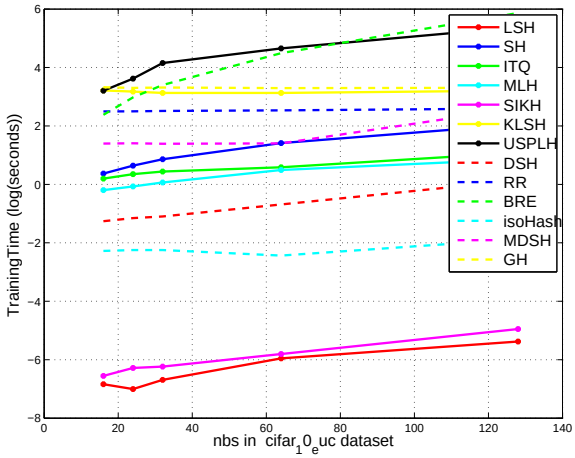
Figure A.8: CIFAR10 Data Precision and Recall Curve at ℓ_2@10% .



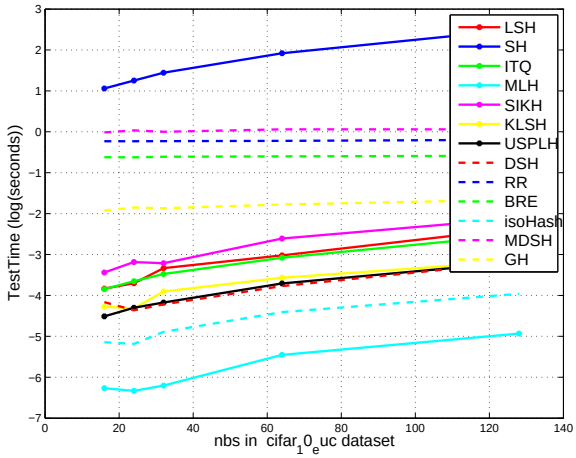
A.9.a: Precision in Hamming radius 3



A.9.b: Area under P-R curve

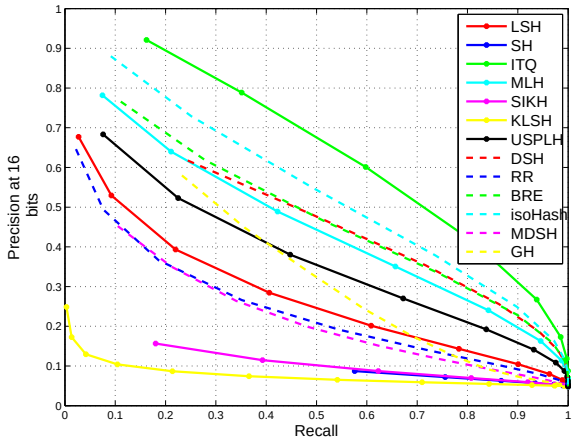


A.9.c: Training Time (log)

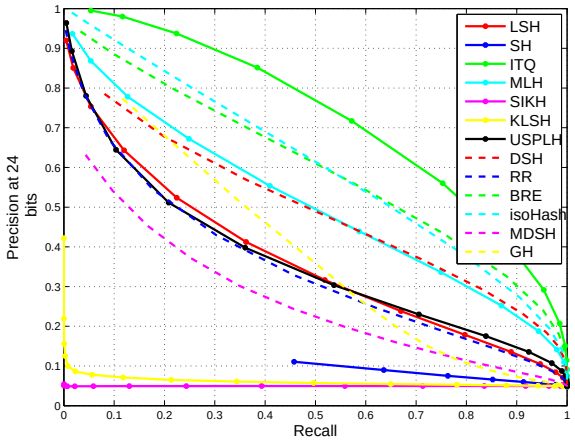


A.9.d: Compress Time (log)

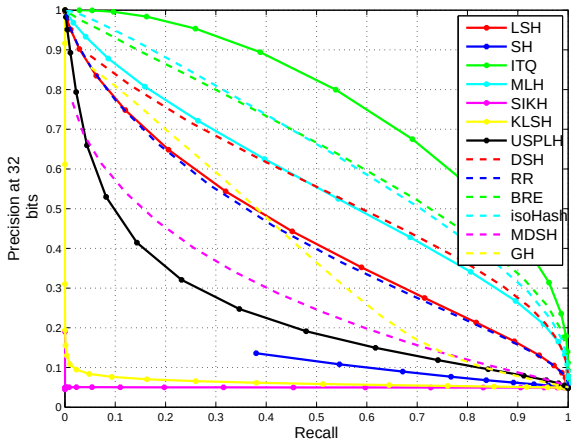
Figure A.9: CIFAR10 Data Precision and Recall Curve at ℓ_2@10%.



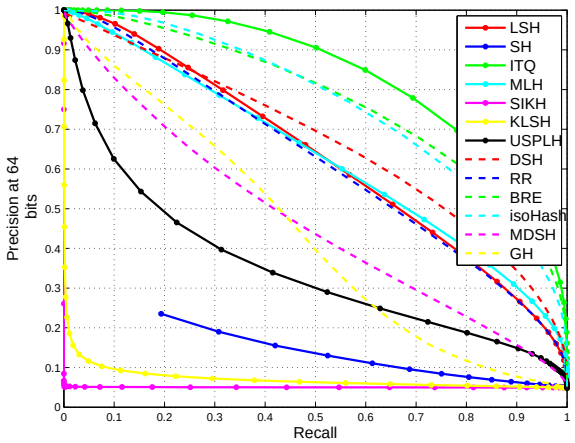
A.10.a: 16 bits



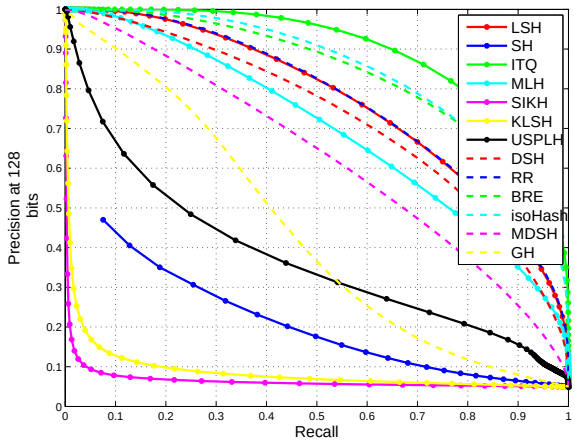
A.10.b: 24 bits



A.10.c: 32 bits

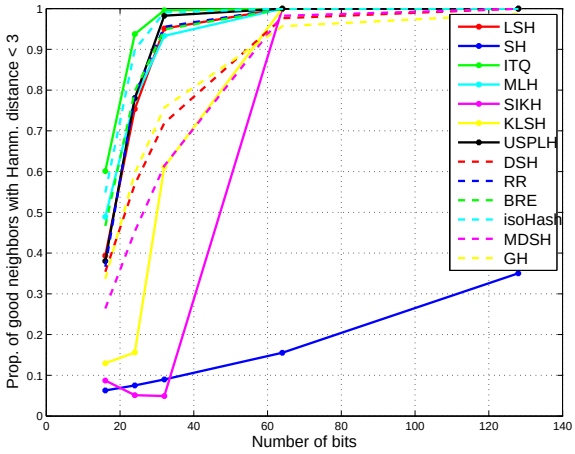


A.10.d: 64 bits

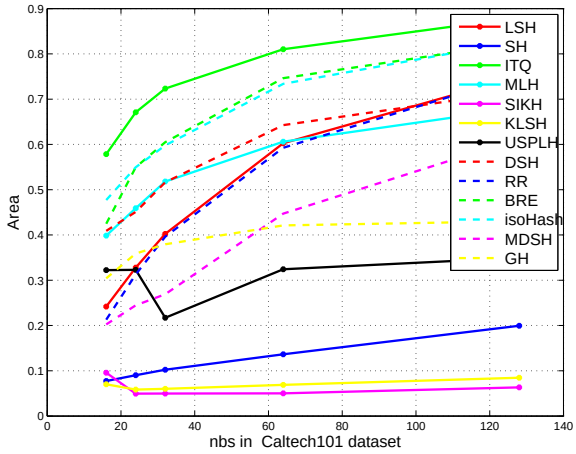


A.10.e: 128 bits

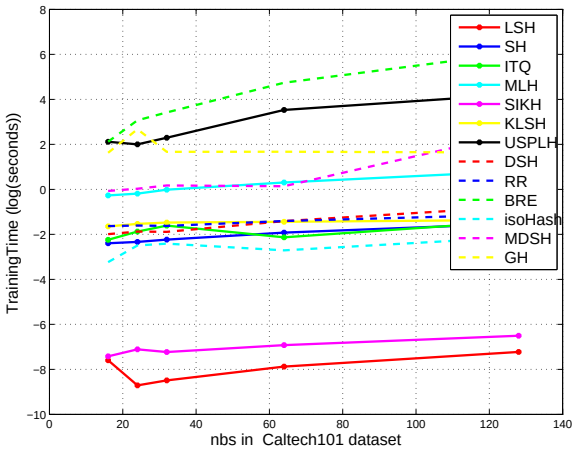
Figure A.10: Caltech101 Data Precision and Recall Curve at ℓ_2@10% .



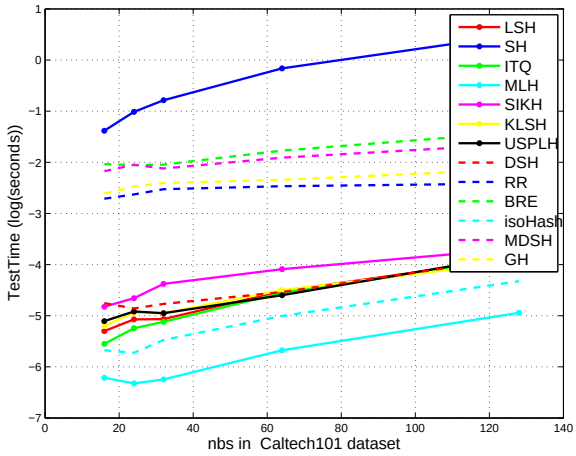
A.11.a: Precision in Hamming radius 3



A.11.b: Area under P-R curve



A.11.c: Training Time (log)



A.11.d: Compress Time (log)

Figure A.11: Caltech101 Data Precision and Recall Curve at ℓ_2@10%.

References

[AHS87] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. Readings in computer vision: issues, problems, principles, and paradigms. chapter A learning algorithm for Boltzmann machines, pages 522–533. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[AI08] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, January 2008.

[AMN+98] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998.

[AWG10] Ryan Prescott Adams, Hanna M. Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. *Journal of Machine Learning Research: Workshop and Conference Proceedings (AISTATS)*, 9:1–8, 05/2010 2010.

[BD09] Yoshua Bengio and Olivier Delalleau. Justifying and generalizing contrastive divergence. *Neural Comput.*, 21(6):1601–1621, June 2009.

[BDLR+04] Yoshua Bengio, Olivier Delalleau, Nicolas Le Roux, Jean-Franccois Paiement, Pascal Vincent, and Marie Ouimet. Learning eigenfunctions links spectral embedding and kernel pca. *Neural Comput.*, 16(10):2197–2219, October 2004.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[Ben09a] Yoshua Bengio. *Learning Deep Architectures for AI*. Now Publishers Inc., Hanover, MA, USA, 2009.

[Ben09b] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. Also published as a book. Now Publishers, 2009.

[BG10] Yoshua Bengio and Xavier Glorot. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS 2010*, volume 9, pages 249–256, May 2010.

[BK] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. Technical Report M217, Philips Research Laboratory, Brussels, Belgium.

[BLBPVtM] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and year = 2012 ee = http://icml.cc/discuss/2012/590.html crossref = DBLP:conf/icml/2012 bibsource = DBLP, http://dblp.uni-trier.de Pascal Vin title = Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription, booktitle = ICML.

[BLP+07] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, UniversitA8A6 De MontrA8A6al, and MontrA8A6al QuA8A6bec. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

[BM] http://en.wikipedia.org/wiki/boltzmann machine.

[BMEM10] Thierry Bertin-Mahieux, Douglas Eck, and Michael I. Mandel. Automatic tagging of audio: The state-of-the-art. In Wenwu Wang, editor, *Machine Audition: Principles, Algorithms and Systems*, chapter 14, pages 334–352. IGI Publishing, 2010.

[Bro97] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–, Washington, DC, USA, 1997. IEEE Computer Society.

[BS97] Anthony Bell and Terrence J. Sejnowski. The 'independent components' of natural scenes are edge filters. *Vision Research*, 37:3327–3338, 1997.

[BSJ11] Bert Huang Blake Shaw and Tony Jebara. Learning a distance metric from a network. In *Advances in Neural Information Processing Systems 24*, 2011.

[CH06] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, September 2006.

[Cha02] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[CKLS09] Kamalika Chaudhuri, Sham M. Kakade, Karen Livescu, and Karthik Sridharan. Multi-view clustering via canonical correlation analysis. In *ICML*, ICML '09, 2009.

[CMP12] Anoop Cherian, Vassilios Morellas, and Nikolaos Papanikolopoulos. Robust sparse hashing. In *ICIP*, pages 2417–2420, 2012.

[CPH05] Miguel A. Carreira-Perpinan and Geoffrey E. Hinton. On contrastive divergence learning, 2005.

[CPZ97] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[CS93] Scott Cost and Steven Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Mach. Learn.*, 10(1):57–78, January 1993.

[CTH+09] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao. Zheng. Nus-wide: A real-world web image database from national university of singapore. In *CIVR*, 2009.

[CW08] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM.

[CXWS12] Minmin Chen, Zhixiang Eddie Xu, Kilian Q. Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. In *ICML*, 2012.

[DAL12] George E. Dahl, Ryan P. Adams, and Hugo Larochelle. Training restricted boltzmann machines on word observations. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 679–686, New York, NY, USA, July 2012. Omnipress.

[DB11] Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. pages 666–674, 2011.

[DDF+90] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.

[DHNZ95] Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. The helmholtz machine, 1995.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, SCG '04, pages 253–262, New York, NY, USA, 2004. ACM.

[DLN] http://deeplearning.net.

[DLS] http://ufldl.stanford.edu.

[DXT12] Lixin Duan, Dong Xu, and Ivor W. Tsang. Learning with augmented features for heterogeneous domain adaptation. In *Proceedings of the International Conference on Machine Learning*, pages 711–718, Edinburgh, Scotland, June 2012. Omnipress.

[EBC+10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660, March 2010.

[Eli05] Saucier-D.M Elias, L.J. *Neuropsychology: Clinical and Experimental Foundations*. Boston: Pearson., 2005.

[EMB+09] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. pages 153–160, April 2009.

[FFFP06] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models for 101 object categories. *CVIU*, 2006.

[FH94] Yoav Freund and David Haussler. Unsupervised learning of distributions on binary vectors using two layer networks, 1994.

[FPSSU96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in knowledge discovery and data mining*. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[FSN+95] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The qbic system. *Computer*, 28(9):23–32, September 1995.

[GCB12] Ian Goodfellow, Aaron Courville, and Yoshua Bengio. Large-scale feature learning with spike-and-slab sparse coding. In *Proc. ICML'2012*, 2012.

[GG91] Allen Gersho and Robert M. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[GKVL12] Yunchao Gong, Sanjiv Kumar, Vishal Verma, and Svetlana Lazebnik. Angular quantization-based binary codes for fast similarity search. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1205–1213. 2012.

[GL11] Yunchao Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 817–824, Washington, DC, USA, 2011. IEEE Computer Society.

[Gra07] Kristen Grauman. Pyramid match hashing: Sub-linear time indexing over partial correspondences. In *In CVPR*, 2007.

[HAYSZ11] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1312–1317. AAAI Press, 2011.

[HDFN95] Geoffrey Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The wake-sleep algorithm for unsupervised neural networks, 1995.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 2012.

[Heb47] Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. 1947.

[HGI00] Taher Haveliwala, Aristides Gionis, and Piotr Indyk. Scalable techniques for clustering the web (extended abstract). In *Third International Workshop on the Web and Databases (WebDB 2000)*, 2000.

[HH02] Patrik O. Hoyer and Aapo Hyvrinen. A multi-layer sparse coding network learns contour coding from natural images, 2002.

[Hin02] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800, 2002.

[Hin07] Geoffrey E. Hinton. Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11(10), 2007.

[Hin10] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. 2010.

[HlBL07] Fu Jie Huang, Y lan Boureau, and Yann Lecun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *In Proc. Computer Vision and Pattern Recognition Conference (CVPR07.* IEEE Press, 2007.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, April 1982.

[HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18:1527–1554, 2006.

[HS83] Geoffrey E. Hinton and Terrence J. Sejnowski. Optimal perceptual inference. In *In CVPR, Washington DC*, 1983.

[HS86] G. E. Hinton and T. J. Sejnowski. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. chapter Learning and relearning in Boltzmann machines, pages 282–317. MIT Press, Cambridge, MA, USA, 1986.

[HS06] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[HSK+12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[HSSSt07] David R. Hardoon, Sandor Szedmak, Or Szedmak, and John Shawe-taylor. Canonical correlation analysis; an overview with application to learning methods. Technical report, 2007.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. pages 604–613, 1998.

[JFY09] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Mach. Learn.*, 77(1):27–59, October 2009.

[JKG08] Prateek Jain, Brian Kulis, and Kristen Grauman. Fast image search for learned metrics. In *CVPR*, 2008.

[KA11] Simon Korman and Shai Avidan. Coherency sensitive hashing. In *ICCV*, pages 1607–1614, 2011.

[Kap95] Hilbert Kappen. Deterministic learning rules for boltzmann machines., 1995.

[KD] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *in Proc. NIPS, 2009*, pages 1042–1050.

[KG09] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision (ICCV*, 2009.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[KL12] Weihao Kong and Wu-Jun Li. Isotropic hashing. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1655–1663. 2012.

[KLG12] Weihao Kong, Wu-Jun Li, and Minyi Guo. Manhattan hashing for large-scale image retrieval. In *SIGIR*, pages 45–54, 2012.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[KU11] Shaishav Kumar and Raghavendra Udupa. Learning hash functions for cross-view similarity search. In *IJCAI*, pages 1360–1365, 2011.

[LBLL09] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *J. Mach. Learn. Res.*, 10:1–40, June 2009.

[LBRN07] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *In NIPS*, pages 801–808. NIPS, 2007.

[LBT10] Hugo Larochelle, Yoshua Bengio, and Joseph Turian. Tractable multivariate binary density estimation and the restricted boltzmann forest. *Neural Comput.*, 22(9):2285–2307, September 2010.

[LCL12] Yue Lin, Deng Cai, and Cheng Li. Density sensitive hashing. *CoRR*, abs/1205.2930, 2012.

[Lee12] Youngwoon Lee. Spherical hashing. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 2957–2964, Washington, DC, USA, 2012. IEEE Computer Society.

[LEN08] Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. Sparse deep belief net model for visual area v2. In *Advances in Neural Information Processing Systems 20*. MIT Press, 2008.

[LEV09] Hugo Larochelle, Dumitru Erhan, and Pascal Vincent. Deep learning using robust interdependent codes. *Journal of Machine Learning Research - Proceedings Track*, 5:312–319, 2009.

[LGRN09] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM.

[LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[LNC+11] Quoc Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Ng. On optimization methods for deep learning. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 265–272, New York, NY, USA, June 2011. ACM.

[LRM+12] Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*, 2012.

[LRY10] Ruei-Sung Lin, David A. Ross, and Jay Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *CVPR*, pages 848–854, 2010.

[LWJ+12] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081, 2012.

[LWKC11] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *ICML*, pages 1–8, 2011.

[MD10] David Andrew Moore and Andrea Pohoreckyj Danyluk. Deep transfer as structure learning in markov logic networks. In *Statistical Relational Artificial Intelligence*, 2010.

[MDG+12] Grégoire Mesnil, Yann Dauphin, Xavier Glorot, Salah Rifai, Yoshua Bengio, Ian J. Goodfellow, Erick Lavoie, Xavier Muller, Guillaume Desjardins, David Warde-Farley, Pascal Vincent, Aaron C. Courville, and James Bergstra. Unsupervised and transfer learning challenge: a deep learning approach. *Journal of Machine Learning Research - Proceedings Track*, 27:97–110, 2012.

[MH08] Andriy Mnih and Geoffrey E. Hinton. A scalable hierarchical distributed language model. In *NIPS*, pages 1081–1088, 2008.

[MH10] Roland Memisevic and Geoffrey E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Comput.*, 22(6):1473–1492, June 2010.

[MNPT05] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Y. Theodoridis. *R-Trees: Theory and Applications (Advanced Information and Knowledge Processing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[MPLB11] Michael Mandel, Razvan Pascanu, Hugo Larochelle, and Yoshua Bengio. Autotagging music with conditional restricted boltzmann machines. March 2011. Online: http://arxiv.org/abs/1103.2832.

[MT12] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1751–1758, 2012.

[NCKN11] Jiquan Ngiam, Zhenghao Chen, Pang Wei Koh, and Andrew Ng. Learning deep energy models. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1105–1112, New York, NY, USA, June 2011. ACM.

[Nea92] Radford M. Neal. Connectionist learning of belief networks. *Artif. Intell.*, 56(1):71–113, July 1992.

[Nea98] Radford M. Neal. Annealed importance sampling. *STATISTICS AND COMPUTING*, 11:125–139, 1998.

[NF11] Mohammad Norouzi and David J. Fleet. Minimal loss hashing for compact binary codes. In *ICML*, pages 353–360, 2011.

[NH09] Vinod Nair and Geoffrey E. Hinton. 3-d object recognition with deep belief nets. In *Advances in Neural Information Processing Systems 22*, 2009.

[NKK+11] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal deep learning. In *International Conference on Machine Learning (ICML)*, Bellevue, USA, June 2011.

[NPF12] Mohammad Norouzi, Ali Punjani, and David J. Fleet. Fast search in hamming space with multi-index hashing. In *CVPR*, pages 3108–3115, 2012.

[PCL06] Christopher Poultney, Sumit Chopra, and Yann Lecun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems (NIPS 2006*. MIT Press, 2006.

[PH01] Alberto Paccanaro and Geoffrey E. Hinton. Learning distributed representations of concepts using linear relational embedding. *IEEE Trans. Knowl. Data Eng.*, 13(2):232–244, 2001.

[QL11] N. Quadrianto and C. Lampert. Learning multi-view neighborhood preserving projections. In *ICML*, 2011.

[RB08] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural Comput.*, 20(6):1631–1649, 2008.

[RBL07] Marc'Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. Sparse feature learning for deep belief networks. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *NIPS*. Curran Associates, Inc., 2007.

[RHW88] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: foundations of research. chapter Learning internal representations by error propagation, pages 673–695. MIT Press, Cambridge, MA, USA, 1988.

[RL09] Maxim Raginsky and Svetlana Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1509–1517. 2009.

[rMDH] Abdel rahman Mohamed, George Dahl, and Geoffrey Hinton. Deep belief networks for phone recognition.

[rMDH12] Abdel rahman Mohamed, George E. Dahl, and Geoffrey E. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech & Language Processing*, 20(1):14–22, 2012.

[RMH10] Marc'Aurelio Ranzato, Volodymyr Mnih, and Geoffrey Hinton. Generating more realistic images using gated mrf's. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2002–2010. 2010.

[RMN09] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 873–880, New York, NY, USA, 2009. ACM.

[RSMH11] Marc'Aurelio Ranzato, Joshua Susskind, Volodymyr Mnih, and Geoffrey Hinton. On deep generative models with applications to recognition. In *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011*. IEEE, 2011.

[SAL12] Jasper Snoek, Ryan Prescott Adams, and Hugo Larochelle. Nonparametric guidance of autoencoder representations using label information. *Journal of Machine Learning Research*, 13:2567–2588, 10/2012 2012.

[Sej86] Terrence J. Sejnowski. Higher-order boltzmann machines. In *Neural Networks for Computing*, pages 398–403. American Institute of Physics, 1986.

[SH] Ruslan Salakhutdinov and Geoffrey Hinton. Replicated softmax: an undirected topic model. In *In Advances in Neural Information Processing Systems*, page 2010.

[SH07] Ruslan Salakhutdinov and Geoffrey Hintondepartment. Using deep belief nets to learn covariance kernels for gaussian processes. In *Advances in Neural Information Processing systems (NIPS-20*, 2007.

[SH09a] Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455, 2009.

[SH09b] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, July 2009.

[SH10] Ruslan Salakhutdinov and Geoffrey Hinton. An efficient learning procedure for deep boltzmann machines, 2010.

[SH12] Ruslan Salakhutdinov and Geoffrey E. Hinton. A better way to pretrain deep boltzmann machines. In *NIPS*, pages 2456–2464, 2012.

[SHT08] Ilya Sutskever, Geoffrey E. Hinton, and Graham W. Taylor. The recurrent temporal restricted boltzmann machine. In *NIPS*, pages 1601–1608, 2008.

[SKC+11] Andrew Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Ng. On random weights and unsupervised feature learning. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1089–1096, New York, NY, USA, June 2011. ACM.

[SM08] Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *ICML*, pages 872–879, 2008.

[SMH11] Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1017–1024, New York, NY, USA, June 2011. ACM.

[Smo86] P. Smolensky. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. chapter Information processing in dynamical systems: foundations of harmony theory, pages 194–281. MIT Press, Cambridge, MA, USA, 1986.

[SOP] Thomas Serre, Aude Oliva, and Tomaso Poggio.

[SS12] Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep boltzmann machines. In *NIPS*, pages 2231–2239, 2012.

[Sto86] G. O. Stone. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. chapter An analysis of the delta rule and the learning of statistical associations, pages 444–459. MIT Press, Cambridge, MA, USA, 1986.

[STS+12] Kevin Swersky, Daniel Tarlow, Ilya Sutskever, Ruslan Salakhutdinov, Richard S. Zemel, and Ryan P. Adams. Cardinality restricted boltzmann machines. In *NIPS*, pages 3302–3310, 2012.

[STT11] Ruslan R. Salakhutdinov, Josh Tenenbaum, and Antonio Torralba. Learning to learn with compound hd models. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2061–2069. 2011.

[SVD03] Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *In ICCV*, pages 750–757, 2003.

[TDR] http://uhavax.hartford.edu/compsci/neural-networks-delta-rule.html.

[TFW08] Antonio Torralba, Robert Fergus, and Yair Weiss. Small codes and large image databases for recognition. In *CVPR'08*, 2008.

[TH09] T. Tieleman and G.E. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th international conference on Machine learning*, pages 1033–1040. ACM New York, NY, USA, 2009.

[Tie08] T. Tieleman. Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM New York, NY, USA, 2008.

[TSH12] Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey Hinton. Deep lambertian networks. In *Proceedings of the 29th International Conference on Machine Learning, 2012, Edinburgh, Scotland*, 2012.

[UK10] Raghavendra Udupa and Mitesh Khapra. Improving the multilingual user experience of wikipedia using cross-language name search. In *HLT '10*, 2010.

[Var07] Manik Varma. Learning the discriminative powerinvariance trade-off. In *In ICCV*, 2007.

[Vin11] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Comput.*, 23(7):1661–1674, July 2011.

[VLBM08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

[WFT12] Yair Weiss, Rob Fergus, and Antonio Torralba. Multidimensional spectral hashing. In *ECCV (5)*, pages 340–353, 2012.

[WKC10a] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, USA, June 2010.

[WKC10b] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential projection learning for hashing with compact codes. In Johannes Frnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 1127–1134. Omnipress, 2010.

[WKC12] Jun Wang, S. Kumar, and Shih-Fu Chang. Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2393–2406, 2012.

[WRZH04] Max Welling, Michal Rosen-Zvi, and Geoffrey E. Hinton. Exponential family harmoniums with an application to information retrieval. In *NIPS*, 2004.

[WS01] Christopher Williams and Matthias Seeger. Using the nystrm method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

[WTF08] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[WYY+14] Fei Wu, Zhou Yu, Yi Yang, Siliang Tang, Yin Zhang, and Yueting Zhuang. Sparse multi-modal hashing. *IEEE Transactions on Multimedia*, 16(2):427–439, 2014.

[XWL+11] Hao Xu, Jingdong Wang, Zhu Li, Gang Zeng, Shipeng Li, and Nenghai Yu. Complementary hashing for approximate nearest neighbor search. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 1631–1638, Washington, DC, USA, 2011. IEEE Computer Society.

[YJ09] Chun-Nam John Yu and Thorsten Joachims. Learning structural svms with latent variables. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1169–1176, New York, NY, USA, 2009. ACM.

[YXG] Kai Yu, Wei Xu, and Yihong Gong. Deep learning with kernel regularization for visual recognition.

[ZBMM06] Hao Zhang, Alexander C. Berg, Michael Maire, and Jitendra Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *In CVPR*, pages 2126–2136, 2006.

[ZHLY08] Jianke Zhu, Steven C.H. Hoi, Michael R. Lyu, and Shuicheng Yan. Near-duplicate keyframe retrieval by nonrigid image matching. In *Proceedings of ACM International Conference on Multimedia (MM2008)*, Vancouver, BC, Canada, October 27–31 2008.

[ZLW+11] Yueting Zhuang, Yang Liu, Fei Wu, Yin Zhang, and Jian Shao. Hypergraph spectral hashing for similarity search of social image. In *ACM Multimedia*, pages 1457–1460, 2011.

[ZMLS07] Jianguo Zhang, Marcin Marszałek, Svetlana Lazebnik, and Cordelia Schmid. Local features and kernels for classification of texture and object categories: a comprehensive study. *International Journal of Computer Vision*, 73(2):213–238, jun 2007.

[ZWCL10] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. Self-taught hashing for fast similarity search. In *SIGIR*, pages 18–25, 2010.

[ZWS11] Dan Zhang, Fei Wang, and Luo Si. Composite hashing with multiple information sources. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 225–234, New York, NY, USA, 2011. ACM.