Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Deep Hashing: A Joint Approach for Image Signature Learning

Yadong Mu,[1] **Zhu Liu**[2]
[1]Institute of Computer Science and Technology, Peking University, China
[2]Multimedia Department, AT&T Labs, U.S.A.
Email: myd@pku.edu.cn, zliu@research.att.com

Abstract

Similarity-based image hashing represents crucial technique for visual data storage reduction and expedited image search. Conventional hashing schemes typically feed hand-crafted features into hash functions, which separates the procedures of feature extraction and hash function learning. In this paper, we propose a novel algorithm that concurrently performs feature engineering and non-linear supervised hashing function learning. Our technical contributions in this paper are twofolds: 1) deep network optimization is often achieved by gradient propagation, which critically requires a smooth objective function. The discrete nature of hash codes makes them not amenable for gradient-based optimization. To address this issue, we propose an exponentiated hashing loss function and its bilinear smooth approximation. Effective gradient calculation and propagation are thereby enabled; 2) pre-training is an important trick in supervised deep learning. The impact of pre-training on the hash code quality has never been discussed in current deep hashing literature. We propose a pre-training scheme inspired by recent advance in deep network based image classification, and experimentally demonstrate its effectiveness. Comprehensive quantitative evaluations are conducted. On all adopted benchmarks, our proposed algorithm generates new performance records by significant improvement margins.

Introduction

Recent years have witnessed spectacular progress on similarity based hash code learning in a variety of computer vision tasks, such as image search (Chum, Philbin, and Zisserman 2008), object recognition (Torralba, Fergus, and Weiss 2008) and local descriptor compression (C. Strecha and Fua 2012) etc. The hash codes are highly compact (*e.g.*, several bytes for each image) in most cases, which significantly reduces the overhead of storing visual big data and also expedites similarity-based image search. The theoretic ground of similarity-oriented hashing is rooted from Johnson-Lindenstrause theorem (Dasgupta and Gupta 2003), which elucidates that for arbitrary n samples, some $\mathcal{O}(\log(n))$-dimensional subspace exists and can be found in polynomial time complexity. When embedded into this subspace, pairwise affinities among these n samples are preserved with tight approximation error bounds. This seminal theoretic discovery sheds light on trading similarity preservation for high compression of large data set. The classic locality-sensitive hashing (LSH) (Indyk and Motwani 1998) is a good demonstration for above tradeoff, instantiated in various similarity metrics such as Hamming distance (Indyk and Motwani 1998), cosine similarity (Charikar 2002), ℓ_p distance with $p \in (0, 2]$ (Datar et al. 2004), and Euclidean distance (Andoni et al. 2013).

Images are often accompanied with supervised information in various forms, such as semantically similar / dissimilar data pairs. Supervised hash code learning (Mu, Shen, and Yan 2010; Wang, Kumar, and Chang 2012) harnesses such supervisory information during parameter optimization and has demonstrated superior image search accuracy compared with unsupervised hashing algorithms (Andoni and Indyk 2008; Weiss, Torralba, and Fergus 2008; Gong et al. 2013). Exemplar supervised hashing schemes include LDAHash (C. Strecha and Fua 2012), two-step hashing (Lin, Shen, and van den Hengel 2015), and kernel-based supervised hashing (Liu et al. 2012) etc. Importantly, two factors crucially affect the accuracy of a supervised hashing algorithm: the discriminative power of the features and the choice of hashing functions. Conventionally, these two factors are separately treated. Images are often represented by hand-crafted visual features (such as SIFT-based bag-of-words feature or sparse codes). Regarding hash functions, a large body of existing works have adopted linear functions owing to the simplicity. More recently, researchers have also explored a number of non-linear hashing functions, such as anchor-based kernalized hashing function (Liu et al. 2012) and decision tree based function (Lin, Shen, and van den Hengel 2015).

This paper attacks supervised hashing by concurrently conducting visual feature engineering and hash function learning. Most of existing image features are designated for generic computer vision tasks. Intuitively, by unifying these two sub-tasks in the same formulation, one can expect the extracted image features to be more amenable for the hashing purpose. Our work is inspired by recent prevalence and success of deep learning techniques (Lecun et al. 1998; Bengio 2009; Krizhevsky, Sutskever, and Hinton 2012). Though the unreasonable effectiveness of deep learning has been successfully demonstrated in tasks like image classification (Krizhevsky, Sutskever, and Hinton 2012) and face

analysis (Sun, Wang, and Tang 2014), deep learning for supervised hashing still remains inadequately explored in the literature.

Xia et al. (Xia et al. 2014) adopted a two-step hashing strategy similar to (Lin, Shen, and van den Hengel 2015). It firstly factorizes the data similarity matrix to obtain the target binary code for each image. In the next stage, the target codes and the image labels are jointly utilized to guide the network parameter optimization. Since the target codes are not updated once approximately learned in the first stage, the final model is only sub-optimal. Lai et al. (Lai et al. 2015) developed a convolutional deep network for hashing, comprised of shared sub-networks and a divide-and-encode module. However, the parameters of these two components are still separately learned. After the shared sub-networks are initialized, their parameters (including all convolutional/pooling layers) are frozen during optimizing the divide-and-encode module. Intrinsically, the method in (Lai et al. 2015) shall be categorized to two-step hashing, rather than simultaneous feature / hashing learning. Liong et al. (Liong et al. 2015) presented a binary encoding network built with purely fully-connected layers. The method essentially assumes that the visual features (*e.g.*, GIST as used in the experiments therein) have been learned elsewhere and fed into its first layer as the input. (Zhang et al. 2015; Zhao et al. 2015) adopt deep networks for learning hash functions in a supervised fashion. However, both methods only support triplets as the source of supervision information, which indicates less efficacy on large data.

The key contributions of this work include: 1) We propose a novel deep hashing algorithm, which takes pairwise similar/dissimilar pairs as inputs and performs concurrent feature and hash function learning over a unified network; 2) We investigate the key pitfalls in designing such deep networks. Particularly, there are two major obstacles: the gradient calculation from non-differentiable binary hash codes, and network pre-training in order to eventually stay at a "good" local optimum. To address the first issue, we propose an exponentiated hashing loss function and devise its bilinear smooth approximation. Effective gradient calculation and propagation are thereby enabled. Moreover, an efficient pre-training scheme is proposed and verified through comprehensive evaluations on real-world visual data.

The proposed method establishes new performance records on four image benchmarks which are widely used in this research area. For instance, on the CIFAR10 dataset, our method achieves a mean average precision of 0.73 for Hamming ranking based image search, which represents some drastic improvement compared with the state-of-the-art methods (0.58 for (Lai et al. 2015) and 0.36 for (Liu et al. 2012)).

The Proposed Method

Throughout this paper we will use bold symbols to denote vectors or matrices, and italic ones for scalars unless otherwise instructed. Suppose a data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with supervision information is provided as the input. Prior works on supervised hashing have considered various forms of supervision, including triplet of items $\langle \mathbf{x}, \mathbf{x}^+, \mathbf{x}^- \rangle$ where the pair $\langle \mathbf{x}, \mathbf{x}^+ \rangle$ is more alike than the pair $\langle \mathbf{x}, \mathbf{x}^- \rangle$ as in (Mu, Shen, and Yan 2010; Lai et al. 2015), pairwise similar/dissimilar relations as in (Liu et al. 2012) or specifying the label of each sample. Observing that triplet-type supervision incurs tremendous complexity during hashing function learning and semantic-level sample labels can be effortlessly converted into pairwise relations, hereafter the discussion focuses on supervision in pairwise fashion. Let $\mathcal{S}$, $\mathcal{D}$ collect all similar / dissimilar pairs respectively. For notational convenience, we further introduce a supervision matrix $\mathbf{Y} \in \{-1, 0, 1\}^{n \times n}$ as

$$\mathbf{Y}_{i,j} = \begin{cases} 1, & (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S} \\ -1, & (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{D} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Figure 1 illustrates our proposed pipeline of learning a deep convolutional network for supervised hashing. The network is comprised of two components: a topmost layer meticulously-customized for the hashing task and other conventional layers. The network takes a $p \times q$-sized images with c channels as the inputs. The K neurons on the top layer output either -1 or 1 as the hash code. Formally, each top neuron represents a hashing function $h_k(\mathbf{x}) : \mathbb{R}^{p \times q \times c} \mapsto \{-1, 1\},\ k = 1 \dots K$, where $\mathbf{x}$ denotes the 3-D raw image. For notational clarity, let us denote the response vector on the second topmost layer as $\mathbf{z} = \phi(\mathbf{x})$, where $\phi(\cdot)$ implicitly defines the highly non-linear mapping from the raw data to a specified intermediate layer.

For the topmost layer, we adopt a simple linear transformation, followed by a signum operation, which is formally presented as

$$h_k(\mathbf{x}) = \text{sign}\left[\mathbf{w}_k^\top \mathbf{z}\right] = \text{sign}\left[\mathbf{w}_k^\top \phi(\mathbf{x})\right]. \tag{2}$$

Exponentiated Code Product Optimization

The key purpose of supervised hashing is to elevate the image search accuracy. The goal can be intuitively achieved by generating discriminative hash codes, such that similar data pairs can be perfectly distinguished from dissimilar pairs according to the Hamming distances calculated over the hash codes. A number of hashing loss functions have been devised by using above design principal. In particular, (Norouzi and Fleet 2011) proposed a hinge-like loss function. Critically, hinge loss is known to be non-smooth and thus complicates gradient-based optimization. Two other works in (Liu et al. 2012; Lin, Shen, and van den Hengel 2015) adopted smooth L_2 loss defined on the inner product between hash codes.

It largely remains unclear for designing optimal hashing loss functions in perceptron-like learning. The major complication stems from the discrete nature of hash codes, which prohibits direct gradient computation and propagation as in typical deep networks. As such, prior works have investigated several tricks to mitigate this issue. Examples include optimizing a variational upper bound of the original non-smooth loss in (Norouzi and Fleet 2011), or simply computing some heuristic-oriented sub-gradients in (Lai et al. 2015). In this work we advocate an exponential discrete loss function which directly optimizes the hash code product and

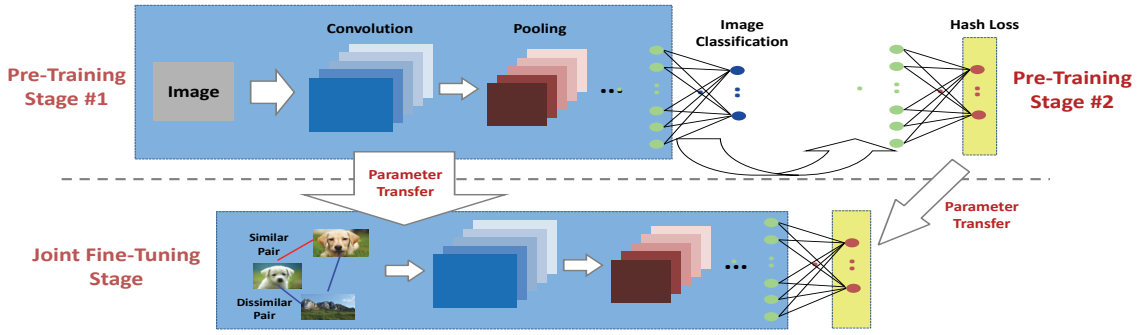

Figure 1: Illustration of our proposed deep network and the pre-training / fine-tuning process. Due to space limit, non-linear activation layers are not plotted in the diagram. See text for more explanations.

enjoys a bilinear smoothed approximation. Compared with other alternative hashing losses, here we first show the proposed exponential loss arguably more amenable for mini-batch based iterative update and later exhibit its empirical superiority in the experiments.

Let $\mathbf{b}_i = \langle h_1(\mathbf{x}_i), \ldots, h_K(\mathbf{x}_i) \rangle^\top \in \{-1, 1\}^K$ denote K hash bits in vector format for data object $\mathbf{x}_i$. We also use the notations $\mathbf{b}_i(k)$, $\mathbf{b}_i(\backslash k)$ to stand for bit k of $\mathbf{b}_i$ and the hash code with bit k absent respectively. As a widely-known fact in the hashing literature (*e.g.*, (Liu et al. 2012)), *code product* admits a one-to-one correspondence to Hamming distance and comparably easier to manipulate. A normalized version of code product ranging over $[-1, 1]$ is described as

$$\mathbf{b}_i \circ \mathbf{b}_j = \frac{1}{K} \sum_{k=1}^{K} \mathbf{b}_i(k)\mathbf{b}_j(k), \tag{3}$$

and when bit k is absent, the code product using partial hash codes is

$$\mathbf{b}_i(\backslash k) \circ \mathbf{b}_j(\backslash k) = \mathbf{b}_i \circ \mathbf{b}_j - \frac{1}{K}\mathbf{b}_i(k)\mathbf{b}_j(k). \tag{4}$$

Exponential Loss: Given the observation that $\mathbf{b}_i \circ \mathbf{b}_j$ faithfully indicates the pairwise similarity, we propose to minimize an exponentiated objective function $\mathcal{Q}$ defined as the accumulation over all data pairs:

$$(\boldsymbol{\theta}^*, \mathbf{w}_k^*) = \arg\min_{\boldsymbol{\theta}, \mathbf{w}_k} \mathcal{Q} \triangleq \sum_{i,j} \ell(\mathbf{x}_i, \mathbf{x}_j), \tag{5}$$

where $\boldsymbol{\theta}$ represents the collection of parameters in the deep networks excluding the hashing loss layer. The atomic loss term is

$$\ell(\mathbf{x}_i, \mathbf{x}_j) = e^{-\mathbf{Y}_{i,j}(\mathbf{b}_i \circ \mathbf{b}_j)}. \tag{6}$$

This novel loss function enjoys some elegant traits desired by deep hashing compared with those in BRE (Kulis and Darrell 2009), MLH (Norouzi and Fleet 2011) and KSH (Liu et al. 2012). It establishes more direct connection to the hashing function parameters by maximizing the correlation of code product and pairwise labeling. In comparison, BRE and MLH optimize the parameters by aligning Hamming distance with original metric distances or enforcing the Hamming distance larger/smaller than pre-specified thresholds. Both formulations incur complicated optimization procedures, and their optimality conditions are unclear. KSH adopts a least-squares formulation for regressing code product onto the target labels, where a smooth surrogate for gradient computation is proposed. However, the surrogate heavily deviates from the original loss function due to its high non-linearity.

Gradient Computation: A prominent advantage of exponential loss is its easy conversion into multiplicative form, which elegantly simplifies the derivation of its gradient. For presentation clarity, we hereafter only focus on the calculation conducted over the topmost hashing loss layer. Namely, $h_k(\mathbf{x}) = \text{sign}\left[\mathbf{w}_k^\top \mathbf{z}\right]$ for bit k, where $\mathbf{z} = \phi(\mathbf{x})$ are the response values at the second top layer and $\mathbf{w}_k$ are parameters to be learned for bit k ($k = 1, \ldots, K$).

Following the common practice in deep learning, two groups of quantities $\partial\mathcal{Q}/\partial\mathbf{w}_k$, $k = 1 \cdots K$ and $\partial\mathcal{Q}/\partial\mathbf{z}_i$ (i ranges over the index set of current mini-batch) need to be estimated on the hashing loss layer at each iteration. The former group of quantities are used for updating $\mathbf{w}_k$, $k = 1 \cdots K$, and the latter are propagated backwards to the bottom layers. The additive algebra of hash code product in Eqn. (3) inspires us to estimate the gradients in a leave-one-out mode. For atomic loss in Eqn. (6), it is easily verified

$$\begin{aligned} \ell(\mathbf{x}_i, \mathbf{x}_j) &= e^{-\mathbf{Y}_{i,j}(\mathbf{b}_i \circ \mathbf{b}_j)} \\ &= e^{-\mathbf{Y}_{i,j}(\mathbf{b}_i(\backslash k) \circ \mathbf{b}_j(\backslash k))} \cdot e^{-\frac{1}{K}\mathbf{Y}_{i,j}(\mathbf{b}_i(k)\mathbf{b}_j(k))}, \end{aligned}$$

where only the latter factor is related to $\mathbf{w}_k$. Since the product $\mathbf{b}_i(k)\mathbf{b}_j(k)$ can only be -1 or 1, we can linearize the latter factor through exhaustively enumerating all possible values, namely

$$e^{-\frac{1}{K}\mathbf{Y}_{i,j}(\mathbf{b}_i(k)\mathbf{b}_j(k))} = c_{i,j} + c'_{i,j} \cdot \Big(\mathbf{b}_i(k)\mathbf{b}_j(k)\Big), \tag{7}$$

where $c_{i,j}, c'_{i,j}$ are two sample-specific constants, calculated by $c_{i,j} = \frac{1}{2}(e^{-\frac{1}{K}\mathbf{Y}_{i,j}} + e^{\frac{1}{K}\mathbf{Y}_{i,j}})$ and $c'_{i,j} = \frac{1}{2}(e^{-\frac{1}{K}\mathbf{Y}_{i,j}} - e^{\frac{1}{K}\mathbf{Y}_{i,j}})$. Since the hardness of calculating the gradient of Eqn. (7) lies in the bit product $\mathbf{b}_i(k)\mathbf{b}_j(k)$, we replace the signum function using the sigmoid-shaped function $\sigma(\mathbf{x}) = 1/(1 + \exp(-x))$, obtaining

$$\begin{aligned} \mathbf{b}_i(k)\mathbf{b}_j(k) &= \text{sign}(\mathbf{w}_k^\top \mathbf{z}_i) \cdot \text{sign}(\mathbf{w}_k^\top \mathbf{z}_j) \\ &= \text{sign}(\mathbf{w}_k^\top \mathbf{z}_i \mathbf{z}_j^\top \mathbf{w}_k) \\ &\approx 2 \cdot \sigma(\mathbf{w}_k^\top \mathbf{z}_i \mathbf{z}_j^\top \mathbf{w}_k) - 1. \end{aligned} \tag{8}$$

Algorithm 1 DeepHash Algorithm

Input: Training set $\mathcal{X}$, data labels, and step size $\eta > 0$;
Output: network parameters $\mathbf{w}_k, k = 1 \cdots K$ for the hashing-loss layer, and $\boldsymbol{\theta}$ for other layers;
pre-training stage #1: initialize $\boldsymbol{\theta}$
Concatenate all layers (excluding top hashing-loss layer) with a softmax layer that defines an image classification task;
Apply AlexNet (Krizhevsky, Sutskever, and Hinton 2012)) style supervised parameter learning algorithm, obtaining $\boldsymbol{\theta}$.
Calculate neuron responses on second topmost layer through $\mathbf{z} = \phi(\mathbf{x}; \boldsymbol{\theta})$;
pre-training stage #2: initialize $\mathbf{w}_k$
Replicate all $\mathbf{z}$'s from previous stage;
while not converged **do**
  Forward computation starting from $\mathbf{z}$;
  for $k = 1$ **to** K **do**
    Update $\mathbf{w}_k$ by minimizing the image classification error;
  end for
end while
simultaneous supervised fine-tuning
while not converged **do**
  Forward computation starting from the raw images;
  for $k = 1$ **to** K **do**
    Estimate $\partial \mathcal{Q}/\partial \mathbf{w}_k \propto \sum_{i,j,k} \partial \ell^{(k)}(\mathbf{z}_i, \mathbf{z}_j)/\partial \mathbf{w}_k$;
    Update $\mathbf{w}_k \leftarrow \mathbf{w}_k - \eta \cdot \partial \mathcal{Q}/\partial \mathbf{w}_k$;
  end for
  Estimate $\partial \mathcal{Q}/\partial \mathbf{z}_i \propto \sum_{j,k} \partial \ell^{(k)}(\mathbf{z}_i, \mathbf{z}_j)/\partial \mathbf{z}_i, \forall i$;
  Propagate $\partial \mathcal{Q}/\partial \mathbf{z}_i$ to bottom layers, updating $\boldsymbol{\theta}$;
end while

Freezing the partial code product $\mathbf{b}_i(\backslash k) \circ \mathbf{b}_j(\backslash k)$, we define an approximate atomic loss with only bits k active:

$$\ell^{(k)}(\mathbf{x}_i, \mathbf{x}_j) \triangleq e^{-\mathbf{Y}_{i,j}(\mathbf{b}_i(\backslash k) \circ \mathbf{b}_j(\backslash k))} \cdot \Big(c_{i,j} + c'_{i,j} \cdot (2 \cdot \sigma(\mathbf{w}_k^\top \mathbf{z}_i \mathbf{z}_j^\top \mathbf{w}_k) - 1) \Big), \quad (9)$$

where the first factor $e^{-\mathbf{Y}_{i,j}(\mathbf{b}_i(\backslash k) \circ \mathbf{b}_j(\backslash k))}$ plays a role of re-weighting specific data pair, conditioned on the rest $K-1$ bits. Iterating over all k's, the original loss function can now be approximated by

$$\ell(\mathbf{x}_i, \mathbf{x}_j) \approx \tfrac{1}{K} \textstyle\sum_{k=1}^{K} \ell^{(k)}(\mathbf{x}_i, \mathbf{x}_j). \quad (10)$$

Compared with other sigmoid-based approximations in previous hashing algorithms (*e.g.*, KSH (Liu et al. 2012)), ours only requires $|\mathbf{w}_k^\top \mathbf{z}_i \mathbf{z}_j^\top \mathbf{w}_k|$ (rather than both $|\mathbf{w}_k^\top \mathbf{z}_i|$ and $|\mathbf{w}_k^\top \mathbf{z}_j|$) is sufficiently large. This bilinearity-oriented relaxation is more favorable for reducing approximation error, which will be corroborated by the subsequent experiments.

Since the objective $\mathcal{Q}$ in Eqn. (5) is a composition of atomic losses on data pairs, we only need to instantiate the gradient computation on specific data pair $(\mathbf{x}_i, \mathbf{x}_j)$. Applying basic calculus rules and discarding some scaling factors, we first obtain

$$\frac{\partial \ell^{(k)}(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{w}_k^\top \mathbf{z}_i \mathbf{z}_j^\top \mathbf{w}_k} \propto e^{-\mathbf{Y}_{i,j}\left(\mathbf{b}_i(\backslash k) \circ \mathbf{b}_j(\backslash k)\right)} \cdot c'_{i,j} \cdot \left(1 - \sigma(\mathbf{w}_k^\top \mathbf{z}_i \mathbf{z}_j^\top \mathbf{w}_k)\right) \cdot \sigma(\mathbf{w}_k^\top \mathbf{z}_i \mathbf{z}_j^\top \mathbf{w}_k),$$

and further using calculus chain rule brings

$$\frac{\partial \ell^{(k)}(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{w}_k} = \frac{\partial \ell^{(k)}(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{w}_k^\top \mathbf{z}_i \mathbf{z}_j^\top \mathbf{w}_k} \cdot \left(\mathbf{z}_i \mathbf{z}_j^\top + \mathbf{z}_j \mathbf{z}_i^\top\right) \mathbf{w}_k,$$

$$\frac{\partial \ell^{(k)}(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{z}_i} = \frac{\partial \ell^{(k)}(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{w}_k^\top \mathbf{z}_i \mathbf{z}_j^\top \mathbf{w}_k} \cdot \left(\mathbf{w}_k \mathbf{w}_k^\top \mathbf{z}_j\right).$$

Importantly, the formulas below obviously hold by the construction of $\ell^{(k)}(\mathbf{x}_i, \mathbf{x}_j)$:

$$\frac{\partial \ell^{(k)}(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{w}_{k'}} = \frac{\partial \ell^{(k)}(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{z}_q} = \mathbf{0}, \; k' \neq k, \; q \neq i, j. \quad (11)$$

Gradient computations on other network layers simply follow the regular calculus rules. We thus omit the introduction.

Two-Stage Supervised Pre-Training

Deep hashing algorithms (including ours) mostly strive to optimize pairwise (or even triplet as in (Lai et al. 2015)) similarity in Hamming space. This raises an intrinsic distinction compared with conventional applications of deep networks (such as image classification via AlexNet (Krizhevsky, Sutskever, and Hinton 2012)). The total count of data pairs quadratically increases with regard to the training sample number, and in conventional applications the number of atomic losses in the objective only linearly grows. This entails a much larger mini-batch size in order to combat numerical instability caused by under-sampling[1], which unfortunately often exceeds the maximal memory space on modern CPU/GPUs.

We adopt a simple two-stage supervised pre-training approach as an effective network pre-conditioner, initializing the parameter values in the appropriate range for further supervised fine-tuning. In the first stage, the network (excluding the hashing loss layer) is concatenated to a regular softmax layer. The network parameters are learned through optimizing the objective of a relevant semantics learning task (*e.g.*, image classification). After stage one is complete, we extract the neuron outputs of all training samples from the second topmost layer (*i.e.*, the variable $\mathbf{z}$'s in Section), feed them into another two-layer shallow network as shown in Figure 1 and initialize the hashing parameters $\mathbf{w}_k$, $k = 1 \cdots K$. Finally, all layers are jointly optimized in a fine-tuning process, minimizing the hashing loss objective $\mathcal{Q}$. The entire procedure is illustrated in Figure 1 and detailed in Algorithm 1.

Experiments

This section reports the quantitative evaluations between our proposed deep hashing algorithm and other competitors.

Description of Datasets: We conduct quantitative comparisons over four image benchmarks which represent different visual classification tasks. They include **MNIST** (Lecun et al. 1998) for handwritten digits recognition, **CIFAR10** (Krizhevsky 2009) which is a subset of *80 million*

[1]For instance, a training set with 100,000 samples demands a mini-batch of 1,000 data for 1% sampling rate in image classification. In contrast, in deep hashing, capturing 1% pairwise similarity requires a tremendous mini-batch of 10,000 data.

Dataset	Train/Query Set	#Class	#Dim	Feature
MNIST	50,000 / 10,000	10	500	CNN
CIFAR10	50,000 / 10,000	10	1,024	CNN
Kaggle-Face	315,799 / 7,178	7	2,304	CNN
SUN397	87,003 / 21,751	397	9,216	CNN

Table 1: Summary of the experimental benchmarks. Feature dimensions correspond to the neurons on the second topmost layer.

Tiny Images dataset and consists of images from ten animal or object categories, **Kaggle-Face**, which is a Kaggle-hosted facial expression classification dataset to stimulate the research on facial feature representation learning, and **SUN397** (Xiao et al. 2010) which is a large scale scene image dataset of 397 categories. For all selected datasets, different classes are completely mutually exclusive such that the similarity/dissimilarity sets as in Eqn. (1) can be calculated purely based on label consensus. Table 1 summarizes the critical information of these experimental data, wherein the column of feature dimension refers to the neurons on the second topmost layers (*i.e.*, dimensions of vector $\mathbf{z}$).

Implementation and Model Specification: We have implemented a substantially-customized version of the open-source Caffe (Jia 2013). The proposed hashing loss layer is patched to the original package and we also largely enrich Caffe's model specification grammar. Moreover, To ensure that mini-batches more faithfully represent the real distribution of pairwise affinities, we re-shuffle the training set at each iteration. This is approximately accomplished by skipping the next few samples (parameterized by a random integer uniformly drawn from $[0, 200]$) in the image database after adding one into the mini-batch. We designate the network layers for each dataset by referring to Caffe's model zoo (Jia 2013). All network configuration information is provided in the supplemental material.

Baselines and Evaluation Protocol: All the evaluations are conducted on a large-scale private cluster, equipped with 12 NVIDIA Tesla K20 GPUs and 8 K40 GPUs. We refer to the proposed algorithm as DeepHash. On the selected benchmarks, DeepHash is compared against classic or state-of-the-art competing hashing schemes, including unsupervised methods such as random projection-based LSH (Charikar 2002), PCAH, SH (Weiss, Torralba, and Fergus 2008), ITQ (Gong et al. 2013), and supervised methods like LDAH (C. Strecha and Fua 2012), MLH (Norouzi and Fleet 2011), BRE (Kulis and Darrell 2009), and KSH (Liu et al. 2012). LSH and PCAH are evaluated using our own implementations. For the rest aforementioned baselines, we thank the authors for publicly sharing their code and adopt the parameters as suggested in the original software packages. Moreover, to make the comparisons comprehensive, four previous deep hashing algorithms are also contrasted, denoted as DH-1 and DH$*$-1 from (Xia et al. 2014), DH-2 (Liong et al. 2015), DH-3 (Lai et al. 2015) and DRSCH (Zhang et al. 2015). Since the authors do not share the source code or model specifications, we instead cite their reported accuracies under identical (or similar) experimental settings.

The performance of a hashing algorithm critically hinges on the semantic discriminatory power of its input features. Previous deep hashing works (Xia et al. 2014; Lai et al. 2015) use traditional hand-crafted features (*e.g.*, GIST and SIFT bag-of-words) for all baselines, which is not an optimal setting for fair comparison with deep hashing. To rule out the effect of less discriminative features, we strictly feed all baselines (except for five deep hashing baseline algorithms) with features extracted from some intermediate layer of the corresponding networks used in deep hashing. Specifically, after the first supervised pre-training stage in Algorithm 1 is completed, we re-arrange the neuron responses on the layer right below the hashing loss layer into vector formats (namely the variable $\mathbf{z}$'s) and feed them into baselines.

All methods share identical training and query sets. After the hashing functions are learned on the training set, all methods produce binary hash codes for the querying data respectively. There exist multiple search strategies using hash codes for image search, such as hash table lookup (Andoni and Indyk 2008) and sparse coding style criterion (Lin, Shen, and van den Hengel 2015). Following recent hashing works, we only carry out Hamming ranking once the hashing functions are learned, which refers to the process of ranking the retrieved samples based on their Hamming distances to the query. Under Hamming ranking protocol, we measure each algorithm using both mean-average-precision (mAP) scores and precision-recall curves.

Investigation of Hamming Ranking Results: Table shows the mAP scores for our proposed DeepHash algorithms (with supervised pre-training and fine-tuning) and all baselines. Due to space limit, we defer more quantitative comparisons (such as the precision-recall curves) to the supplemental material. There are three key observations from these experimental results that we would highlight:

1) On all datasets, our proposed DeepHash algorithm significantly perform better than all baselines in terms of mAP. For all non-deep-network based algorithm, KSH achieves the best accuracies on MNIST, CIFAR10 and Kaggle-Face, and ITQ shows top performances on SUN397. Using 48 hash bits, the best mAP scores obtained by KSH or ITQ are 0.9817, 0.5482, 0.4132, and 0.0471 on MNIST / CIFAR10 / Kaggle-Face / SUN397 respectively. In comparison, our proposed DeepHash performs nearly perfect on MNIST (0.9938), and defeat KSH and ITQ by very large margins, scoring 0.7410, 0.5615, and 0.1293 on other three datasets respectively.

2) We also include five deep hashing algorithms by referring to the accuracies reported in the original publications. Recall that the evaluations in (Xia et al. 2014; Lai et al. 2015) feed baseline algorithms with non-CNN features (*e.g.*, GIST). Interestingly, our experiments reveal that, when conventional hashing algorithms take CNN features as the input, the relative performance gain of prior deep hashing algorithms becomes marginal. For example, under 48 hash bits, KSH's mAP score 0.5482 is comparable with regard to DH-3's 0.581. We attribute the striking superiority of our proposed deep hashing algorithm to the importance of jointly conducting feature engineering and hash function

	MNIST			CIFAR10			Kaggle-Face			SUN397		
	12 bits	24 bits	48 bits	12 bits	24 bits	48 bits	12 bits	24 bits	48 bits	12 bits	24 bits	48 bits
LSH	0.3717	0.4933	0.5725	0.1311	0.1619	0.2034	0.1911	0.2011	0.1976	0.0057	0.0060	0.0071
ITQ	0.7578	0.8132	0.8293	0.2711	0.2825	0.2909	0.2435	0.2513	0.2514	0.0268	0.0361	0.0471
PCAH	0.4997	0.4607	0.3641	0.2056	0.1867	0.1695	0.2169	0.2058	0.1991	0.0218	0.0261	0.0315
SH	0.5175	0.5330	0.4898	0.1935	0.1921	0.1750	0.2117	0.2054	0.2015	0.0210	0.0236	0.0273
LDAH	0.5052	0.3685	0.3093	0.2187	0.1794	0.1587	0.2154	0.2032	0.1961	0.0224	0.0262	0.0306
BRE	0.6950	0.7498	0.7785	0.2552	0.2668	0.2864	0.2414	0.2522	0.2587	0.0226	0.0293	0.0372
MLH	0.6731	0.4404	0.4258	0.1737	0.1675	0.1737	0.2000	0.2115	0.2162	0.0070	0.0100	0.0210
KSH	0.9537	0.9713	0.9817	0.3441	0.4617	0.5482	0.2862	0.3668	0.4132	0.0194	0.0261	0.0325
DH-1	0.957	0.963	0.960	0.439	0.511	0.522	–	–	–	–	–	–
DH$*$-1	0.969	0.975	0.975	0.465	0.521	0.532	–	–	–	–	–	–
DH-2	0.4675	0.5101	0.5250	0.1880	0.2083	0.2251	–	–	–	–	–	–
DH-3	–	–	–	0.552	0.566	0.581	–	–	–	–	–	–
DRSCH	0.9692	0.9737	0.9791	0.6246	0.6219	0.6305	–	–	–	–	–	–
DeepHash	**0.9918**	**0.9931**	**0.9938**	**0.6874**	**0.7289**	**0.7410**	**0.5487**	**0.5552**	**0.5615**	**0.0748**	**0.1054**	**0.1293**

Table 2: Experimental results in terms of mean-average-precision (mAP) under various hash bits. The mAP scores are calculated based on Hamming ranking. Best scores are highlighted in bold. Note that the mAP scores are in the numerical range of $[0, 1]$. We directly cite the performance In the table, "–" indicates the corresponding scores are not available. Refer to text for more details.

	MNIST			CIFAR10			Kaggle-Face			SUN397		
	12 bits	24 bits	48 bits	12 bits	24 bits	48 bits	12 bits	24 bits	48 bits	12 bits	24 bits	48 bits
random init.	0.9806	0.9862	0.9873	0.5728	0.6503	0.6585	0.4125	0.4473	0.4620	0.0211	0.0384	0.0360
pre-training	0.9673	0.9753	0.9796	0.4986	0.5588	0.5966	0.4282	0.4484	0.4589	0.0335	0.0430	0.0592
fine-tuning	0.9918	0.9931	0.9938	0.6874	0.7289	0.7410	0.5487	0.5552	0.5615	0.0748	0.1054	0.1293

Table 3: Comparisons of three strategies of parameter initialization and learning for the proposed DeepHash. See text for more details.

learning (*i.e.*, the fine-tuning process in Algorithm 1).

3) Elevating inter-bit mutual complementarity is overly crucial for the final performance. For those methods that generate hash bits independently (such as LSH) or by enforcing performance-irrelevant inter-bit constraints (such as LDAH), the mAP scores only show slight gains or even drop when increasing hash code length. Among all algorithms, two code-product oriented algorithm, KSH and our proposed DeepHash, show steady improvement by using more hash bits. Moreover, our results also validate some known insights exposed by previous works, such as the advantage of supervised hashing methods over the unsupervised alternatives.

Effect of Supervised Pre-Training: We now further highlight the effectiveness of the two-stage supervised pre-training process. To this end, in Table we show the mAP scores achieved by three different strategies of learning the network parameters. The scheme "random init." refers to initializing all parameters with random numbers without any pre-training. A typical supervised gradient back-propagation procedure as in AlexNet (Krizhevsky, Sutskever, and Hinton 2012) is then used. The second scheme "pre-training" refers to initializing the network using two-stage pre-training in Algorithm 1, without any subsequent fine-tuning. It serves as an appropriate baseline for assessing the benefit of the fine-tuning process as in the third scheme "fine-tuning". In all cases, the learning rate in gradient descent drops at a constant factor (0.1 in all of our experiments) until the training converges.

There are two major observations from Table . First, simultaneous tuning all the layers (including the hashing loss layer) often significantly boosts the performance. As a key evidence, "random init." demonstrates prominent superiority on MNIST and CIFAR10 compared with "pre-training". The joint parameter tuning of "random init." is supposed to compensate the low-quality random parameter initialization. Secondly, positioning the initial solution near a "good" local optimum is crucial for learning on challenging data. For example, the dataset of SUN397 has as many as 397 unique scene categories. However, due to the limitation of GPU memory, even a K40 GPU with 12GB memory only support a mini-batch of 600 samples at maximum. State differently, each mini-batch only comprises 1.5 samples per category on average, which results in a heavily biased sampling. We attribute the relatively low accuracies of "random init." to this issue. In contrast, training deep networks with supervised pre-training and fine-tuning (*i.e.*, the third scheme in Table) exhibit robust performances over all datasets.

Concluding Remarks

We accredit the success of deep hashing to the joint feature / hash function learning, and a novel exponential loss function whose approximation in Eqn. (10) excellently fits the mini-batch based optimization. To combat the under-

sampling issue during training, we introduce two-stage supervised pre-training and validate its effectiveness by comparisons. Our comprehensive quantitative evaluations consistently demonstrate the power of deep hashing for the data hashing task. The proposed algorithm enjoys both scalability to large training data and millisecond-level testing time for processing a new image. We thus believe that deep hashing is promising for efficiently analyzing visual big data.

References

Andoni, A., and Indyk, P. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM* 51(1):117–122.

Andoni, A.; Indyk, P.; Nguyen, H. L.; and Razenshteyn, I. 2013. Beyond locality-sensitive hashing. *CoRR* abs/1306.1547.

Bengio, Y. 2009. Learning deep architectures for AI. *Found. Trends Mach. Learn.* 2(1):1–127.

C. Strecha, A. M. Bronstein, M. M. B., and Fua, P. 2012. LDAHash: improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34(1).

Charikar, M. 2002. Similarity estimation techniques from rounding algorithms. In *STOC*.

Chum, O.; Philbin, J.; and Zisserman, A. 2008. Near duplicate image detection: min-hash and tf-idf weighting. In *BMVC*.

Dasgupta, S., and Gupta, A. 2003. An elementary proof of a theorem of johnson and lindenstrauss. *Random Struct. Algorithms* 22(1):60–65.

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *STOC*.

Gong, Y.; Lazebnik, S.; Gordo, A.; and Perronnin, F. 2013. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.* 35(12):2916–2929.

Indyk, P., and Motwani, R. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*.

Jia, Y. 2013. Caffe: An open source convolutional architecture for fast feature embedding. http://caffe.berkeleyvision.org/.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report.

Kulis, B., and Darrell, T. 2009. Learning to hash with binary reconstructive embeddings. In *NIPS*.

Lai, H.; Pan, Y.; Liu, Y.; and Yan, S. 2015. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 2278–2324.

Lin, G.; Shen, C.; and van den Hengel, A. 2015. Supervised hashing using graph cuts and boosted decision trees. *IEEE Trans. Pattern Anal. Mach. Intell.* 37(11):2317–2331.

Liong, V. E.; Lu, J.; Wang, G.; Moulin, P.; and Zhou, J. 2015. Deep hashing for compact binary codes learning. In *CVPR*.

Liu, W.; Wang, J.; Ji, R.; Jiang, Y.; and Chang, S. 2012. Supervised hashing with kernels. In *CVPR*.

Mu, Y.; Shen, J.; and Yan, S. 2010. Weakly-supervised hashing in kernel space. In *CVPR*.

Norouzi, M., and Fleet, D. J. 2011. Minimal loss hashing for compact binary codes. In *ICML*.

Sun, Y.; Wang, X.; and Tang, X. 2014. Deep learning face representation from predicting 10, 000 classes. In *CVPR*.

Torralba, A.; Fergus, R.; and Weiss, Y. 2008. Small codes and large image databases for recognition. In *CVPR*.

Wang, J.; Kumar, S.; and Chang, S.-F. 2012. Semi-supervised hashing for large-scale search. *IEEE Trans. Pattern Anal. Mach. Intell.* 34(12):2393–2406.

Weiss, Y.; Torralba, A.; and Fergus, R. 2008. Spectral hashing. In *NIPS*.

Xia, R.; Pan, Y.; Lai, H.; Liu, C.; and Yan, S. 2014. Supervised hashing for image retrieval via image representation learning. In *AAAI*.

Xiao, J.; Hays, J.; Ehinger, K. A.; Oliva, A.; and Torralba, A. 2010. SUN database: Large-scale scene recognition from abbey to zoo. In *CVPR*.

Zhang, R.; Lin, L.; Zhang, R.; Zuo, W.; and Zhang, L. 2015. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *IEEE Transactions on Image Processing* 24(12):4766–4779.

Zhao, F.; Huang, Y.; Wang, L.; and Tan, T. 2015. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*.