# Efficient end-to-end learning for quantizable representations

**Yeonwoo Jeong** [1]  **Hyun Oh Song** [1]

## Abstract

Embedding representation learning via neural networks is at the core foundation of modern similarity based search. While much effort has been put in developing algorithms for learning binary hamming code representations for search efficiency, this still requires a linear scan of the entire dataset per each query and trades off the search accuracy through binarization. To this end, we consider the problem of directly learning a quantizable embedding representation and the sparse binary hash code end-to-end which can be used to construct an efficient hash table not only providing significant search reduction in the number of data but also achieving the state of the art search accuracy outperforming previous state of the art deep metric learning methods. We also show that finding the optimal sparse binary hash code in a mini-batch can be computed exactly in polynomial time by solving a *minimum cost flow* problem. Our results on Cifar-100 and on ImageNet datasets show the state of the art search accuracy in precision@k and NMI metrics while providing up to $98\times$ and $478\times$ search speedup respectively over exhaustive linear search. The source code is available at https://github.com/maestrojeong/Deep-Hash-Table-ICML18.

## 1. Introduction

Learning the representations that respect the pairwise relationships is one of the most important problems in machine learning and pattern recognition with vast applications. To this end, deep metric learning methods (Hadsell et al., 2006; Weinberger et al., 2006; Schroff et al., 2015; Song et al., 2016; Sohn, 2016; Song et al., 2017; Bell & Bala, 2015; Sener et al., 2016) aim to learn an embedding representation space such that similar data are close to each other

and vice versa for dissimilar data. Some of these methods have shown significant advances in various applications in retrieval (Sohn, 2016; Schroff et al., 2015), clustering (Song et al., 2017), domain adaptation (Sener et al., 2016), video understanding (Wang & Gupta, 2015), etc.

Despite recent advances in deep metric learning methods, deploying the learned embedding representation in large scale applications poses great challenges in terms of the inference efficiency and scalability. To address this, practitioners in large scale retrieval and recommendation systems often resort to a separate post-processing step where the learned embedding representation is run through quantization pipelines such as sketches, hashing, and vector quantization in order to significantly reduce the number of data to compare during the inference while trading off the accuracy.

In this regard, we propose a novel end-to-end learning method for quantizable representations jointly optimizing for the quality of the network embedding representation and the performance of the corresponding binary hash code. In contrast to some of the recent methods (Cao et al., 2016; Liu & Lu, 2017), our proposed method avoids ever having to cluster the entire dataset, offers the modularity to accommodate any existing deep embedding learning techniques (Schroff et al., 2015; Sohn, 2016), and is efficiently trained in a mini-batch stochastic gradient descent setting. We show that the discrete optimization problem of finding the optimal binary hash codes given the embedding representations can be computed efficiently and exactly by solving an equivalent *minimum cost flow problem*. The proposed method alternates between finding the optimal hash codes of the given embedding representations in the mini-batch and adjusting the embedding representations indexed at the activated hash code dimensions via deep metric learning methods.

Our end-to-end learning method outperforms state-of-the-art deep metric learning approaches (Schroff et al., 2015; Sohn, 2016) in retrieval and clustering tasks on the Cifar-100 (Krizhevsky et al., 2009) and the ImageNet (Russakovsky et al., 2015) datasets while providing up to several orders of magnitude speedup during inference. Our method utilizes efficient off-the-shelf implementations from OR-Tools (Google optimization tools for combinatorial optimization problems) (OR-tools, 2018), the deep metric learning library implementation in Tensorflow (Abadi et al., 2015),

---

[1]Department of Computer Science and Engineering, Seoul National University, Seoul, Korea. Correspondence to: Hyun Oh Song <hyunoh@snu.ac.kr>.

and is efficient to train. The state of the art deep metric learning approaches (Schroff et al., 2015; Sohn, 2016) use the class labels during training (for the hard negative mining procedure) and since our method utilizes the approaches as a component, we focus on the settings where the class labels are available during training.

## 2. Related works

Learning the embedding representation via neural networks dates back to over two decades ago. Starting with Siamese networks (Bromley et al., 1994; Hadsell et al., 2006), the task is to learn embedding representation of the data using neural networks so that similar examples are close to each other and dissimilar examples are farther apart in the embedding space. Despite the recent successes and near human performance reported in retrieval and verification tasks (Schroff et al., 2015), most of the related literature on learning efficient representation via deep learning focus on learning the binary hamming codes for finding nearest neighbors with linear search over entire dataset per each query. Directly optimizing for the embedding representations in deep networks for quantization codes and constructing the hash tables for significant search reduction in the number of data is much less studied.

(Xia et al., 2014) first precompute the hash code based on the labels and trains the embedding to be similar to the hash code. (Zhao et al., 2015) apply element-wise sigmoid on the embedding and minimizes the triplet loss. (Norouzi et al., 2012) optimize the upper bound on the triplet loss defined on hamming code vectors. (Liong et al., 2015) minimize the difference between the original and the signed version of the embedding with orthogonality regularizers in a network. (Li et al., 2017) employ discrete cyclic coordinate descent (Shen et al., 2015) on a discrete sub-problem optimizing one hash bit at a time but the algorithm has neither the convergence guarantees nor the bound on the number of iterations. All of these methods focus on learning the binary codes for hamming distance ranking and perform an exhaustive linear search over the entire dataset which is not likely to be suitable for large scale problems.

(Cao et al., 2016) minimize the difference between the similarity label and the cosine distance of network embedding. (Liu & Lu, 2017) define a distance between a quantized data and continuous embedding, and back-propagates the metric loss error only with respect to the continuous embedding. Both of these methods require *repeatedly* running k-means clustering on the *entire* dataset while training the network at the same time. This is unlikely to be practical for large scale problems because of the prohibitive computational complexity and having to store the cluster centroids for all classes in the memory as the number of classes becomes extremely large (Prabhu & Varma, 2014; Choromanska et al., 2013).

In this paper, we propose an efficient end-to-end learning method for quantizable representations jointly optimizing the quality of the embedding representation and the performance of the corresponding hash codes in a scalable mini-batch stochastic gradient descent setting in a deep network and demonstrate state of the art search accuracy and quantitative search efficiency on multiple datasets.

## 3. Problem formulation

Consider a hash function $r(\mathbf{x})$ that maps an input data $\mathbf{x} \in \mathcal{X}$ onto a $d$ dimensional binary compound hash code $\mathbf{h} \in \{0,1\}^d$ with the constraint that $k$ out of $d$ total bits needs to be set. We parameterize the mapping as

$$r(\mathbf{x}) = \underset{\mathbf{h} \in \{0,1\}^d}{\operatorname{argmin}} -f(\mathbf{x}; \boldsymbol{\theta})^{\mathsf{T}} \mathbf{h}$$
$$\text{subject to } ||\mathbf{h}||_1 = k, \tag{1}$$

where $f(\cdot, \boldsymbol{\theta}) : \mathcal{X} \to \mathbb{R}^d$ is a transformation (i.e. neural network) differentiable with respect to the parameter $\boldsymbol{\theta}$ and takes the input $\mathbf{x}$ and emits the $d$ dimensional embedding vector. Given the hash function $r(\cdot)$, we define a hash table $H$ which is composed of $d$ buckets with each bucket indexed by a compound hash code $\mathbf{h}$. Then, given a query $\mathbf{x}_q$, union of all the items in the buckets indexed by $k$ active bits in $r(\mathbf{x}_q)$ is retrieved as the candidates of the approximate nearest items of $\mathbf{x}_q$. Finally, this is followed by a reranking operation where the retrieved items are ranked according to the distances computed using the original embedding representation $f(\cdot; \boldsymbol{\theta})$.

Note, in quantization based hashing (Wang et al., 2016; Cao et al., 2016), a set of prototypes or cluster centroids are first computed via dictionary learning or other clustering (i.e. k-means) algorithms. Then, the function $f(\mathbf{x}; \boldsymbol{\theta})$ is represented by the indices of k-nearest prototypes or centroids. Concretely, if we replace $f(\mathbf{x}; \boldsymbol{\theta})$ in Equation (1) with the negative distances of the input item $\mathbf{x}$ with respect to all $d$ prototypes or centroids, $[-||\mathbf{x} - \mathbf{c}_1||_2, \ldots, -||\mathbf{x} - \mathbf{c}_d||_2]^{\mathsf{T}}$, then the corresponding hash function $r(\mathbf{x})$ can be used to build the hash table.

In contrast to most of the recent methods that learn a hamming ranking in a neural network and perform exhaustive linear search over the entire dataset (Xia et al., 2014; Zhao et al., 2015; Norouzi et al., 2012; Li et al., 2017), quantization based methods, have guaranteed search inference speed up by only considering a subset of $k$ out $d$ buckets and thus avoid exhaustive linear search. We explicitly maintain the sparsity constraint on the hash code in Equation (1) throughout our optimization without continuous relaxations to inherit the efficiency aspect of quantization based hashing and this is one of the key attributes of the algorithm.

Although quantization based hashing is known to show high search accuracy and search efficiency (Wang et al., 2016),

running the quantization procedure on the entire dataset to compute the cluster centroids is computationally very costly and requires storing all of the cluster centroids in the memory. Our desiderata are to formulate an efficient end-to-end learning method for quantizable representations which (1) guarantees the search efficiency by avoiding linear search over the entire data, (2) can be efficiently trained in a mini-batch stochastic gradient descent setting and avoid having to quantize the entire dataset or having to store the cluster centroids for all classes in the memory, and (3) offers the modularity to accommodate existing embedding representation learning methods which are known to show the state of the art performance on retrieval and clustering tasks.

# 4. Methods

We formalize our proposed method in Section 4.1 and discuss the subproblems in Section 4.2 and in Section 4.4.

## 4.1. End-to-end learning for quantizable representations

Finding the optimal set of embedding representations and the corresponding hash codes is a chicken and egg problem. Embedding representations are required to infer which $k$ activation dimensions to set in the corresponding binary hash code, but the binary hash codes are needed to adjust the embedding representations indexed at the activated bits so that similar items get hashed to the same buckets and vice versa. We formalize this notion in Equation (2) below.

$$\underset{\substack{\boldsymbol{\theta} \\ \mathbf{h}_1,\ldots,\mathbf{h}_n}}{\text{minimize}} \quad \underbrace{\ell_{\text{metric}}(\{f(\mathbf{x}_i;\boldsymbol{\theta})\}_{i=1}^n; \mathbf{h}_1,\ldots,\mathbf{h}_n)}_{\text{embedding representation quality}} +$$
$$\gamma \underbrace{\left( \sum_i^n -f(\mathbf{x}_i;\boldsymbol{\theta})^\mathsf{T}\mathbf{h}_i + \sum_i^n \sum_{j:y_j\neq y_i} \mathbf{h}_i^\mathsf{T} P \mathbf{h}_j \right)}_{\text{hash code performance}}$$
$$\text{subject to} \quad \mathbf{h}_i \in \{0,1\}^d, ||\mathbf{h}_i||_1 = k, \, \forall i, \tag{2}$$

where the matrix $P$ encodes the pairwise cost for the hash code similarity between each negative pair and $\gamma$ is the trade-off hyperparameter balancing the loss contribution between the embedding representation quality given the hash codes and the performance of the hash code with respect to the embedding representations. We solve this optimization problem via alternating minimization through iterating over solving for $k$-sparse binary hash codes $\mathbf{h}_1,\ldots,\mathbf{h}_n$ and updating the parameters of the deep network $\boldsymbol{\theta}$ for the continuous embedding representations per each mini-batch. Following subsections discuss these two steps in detail.

## 4.2. Learning the compound hash code

Given a set of continuous embedding representations $\{f(\mathbf{x}_i;\boldsymbol{\theta})\}_{i=1}^n$, we seek to solve the following subproblem in Equation (3) where the task is to (*unary*) select $k$ as large elements of the each embedding vector as possible, while (*pairwise*) selecting as orthogonal elements as pos-

sible across different classes. The unary term mimics the hash function $r(\mathbf{x})$ in Equation (1) and the pairwise term has the added benefit that it also provides robustness to the optimization especially during the early stages of the training when the embedding representation is not very accurate.

$$\underset{\mathbf{h}_1,\ldots,\mathbf{h}_n}{\text{minimize}} \quad \underbrace{\sum_i^n -f(\mathbf{x}_i;\boldsymbol{\theta})^\mathsf{T}\mathbf{h}_i + \sum_i^n \sum_{j:y_j\neq y_i} \mathbf{h}_i^\mathsf{T} P \mathbf{h}_j}_{:=g(\mathbf{h}_{1,\ldots,n};\boldsymbol{\theta})}$$
$$\text{subject to} \quad \mathbf{h}_i \in \{0,1\}^d, ||\mathbf{h}_i||_1 = k, \, \forall i, \tag{3}$$

However, solving for the optimal solution of the problem in Equation (3) is NP-hard in general even for the simple case where $k = 1$ and $d > 2$ (Boykov et al., 2001). Thus, we construct a upper bound function $\bar{g}(\mathbf{h}_{1,\ldots,n};\boldsymbol{\theta})$ to the objective function $g(\mathbf{h}_{1,\ldots,n};\boldsymbol{\theta})$ which we argue that it can be exactly optimized by establishing the connection to a network flow algorithm. The upper bound function is a slightly reparameterized discrete objective where we optimize the hash codes over the average embedding vectors per each class instead. We first rewrite[1] the objective function by indexing over each class and then over each data per class and derive the upper bound function as shown below.

$$g(\mathbf{h}_{1,\ldots,n};\boldsymbol{\theta}) = \sum_i^{n_c} \sum_{k:y_k=i} -f(\mathbf{x}_k;\boldsymbol{\theta})^\mathsf{T}\mathbf{h}_k + \sum_i^{n_c} \sum_{\substack{k:y_k=i,\\l:y_l\neq i}} \mathbf{h}_k^\mathsf{T} P \mathbf{h}_l$$

$$\leq \sum_i^{n_c} \sum_{k:y_k=i} -\mathbf{c}_i^\mathsf{T}\mathbf{h}_k + \sum_i^{n_c} \sum_{\substack{k:y_k=i,\\l:y_l\neq i}} \mathbf{h}_k^\mathsf{T} P \mathbf{h}_l$$

$$+ \underbrace{\underset{\substack{\mathbf{h}_1,\ldots,\mathbf{h}_n \\ \mathbf{h}_i\in\{0,1\}^d,||\mathbf{h}_i||_1=k}}{\text{maximize}} \sum_{i=1}^{n_c} \sum_{k:y_k=i} (\mathbf{c}_i - f(\mathbf{x}_k;\boldsymbol{\theta}))^\mathsf{T}\mathbf{h}_k}_{:=M(\boldsymbol{\theta})}$$

$$= \bar{g}(\mathbf{h}_{1,\ldots,n};\boldsymbol{\theta}) \tag{4}$$

where $n_c$ denotes the number of classes in the mini-batch, $m = |\{k : y_k = i\}|$, and $\mathbf{c}_i = \frac{1}{m}\sum_{k:y_k=i} f(\mathbf{x}_k;\boldsymbol{\theta})$. Here, w.l.o.g we assume each class has $m$ number of data in the mini-batch (i.e. Npairs (Sohn, 2016) mini-batch construction). The last term in upper bound, denoted as $M(\boldsymbol{\theta})$, is constant with respect to the hash codes and is non-negative. Note, from the bound in Equation (4), the gap between the minimum value of $g$ and the minimum value of $\bar{g}$ is bounded above by $M(\boldsymbol{\theta})$. Furthermore, since this value corresponds to the maximum deviation of an embedding vector from its class mean of the embedding, the bound gap decreases over iterations as we update the network parameter $\boldsymbol{\theta}$ to attract similar pairs of data and vice versa for dissimilar pairs in the other embedding subproblem (more details in Section 4.4).

Moreover, minimizing the upper bound over each hash codes $\{\mathbf{h}_i\}_{i=1}^n$ is equivalent to minimizing a reparameter-

---

[1]We also omit the dependence of the index $i$ for each $\mathbf{h}_k$ and $\mathbf{h}_l$ to avoid the notation clutter.

ization $\hat{g}(\mathbf{z}_{1,\dots,n_c}; \boldsymbol{\theta})$ over $\{\mathbf{z}_i\}_{i=1}^{n_c}$ defined below because for a given class label $i$, each $\mathbf{h}_k$ shares the same $\mathbf{c}_i$ vector.

$$
\begin{aligned}
& \underset{\substack{\mathbf{h}_1,\dots,\mathbf{h}_n \\ \mathbf{h}_i \in \{0,1\}^d, ||\mathbf{h}_i||_1 = k}}{\text{minimize}} \quad \sum_i^{n_c} \sum_{k:y_k=i} -\mathbf{c}_i^\mathsf{T}\mathbf{h}_k + \sum_i^{n_c} \sum_{\substack{k:y_k=i, \\ l:y_l \neq i}} \mathbf{h}_k^\mathsf{T} P \mathbf{h}_l \\
& = \underset{\substack{\mathbf{z}_1,\dots,\mathbf{z}_{n_c} \\ \mathbf{z}_i \in \{0,1\}^d, ||\mathbf{z}_i||_1=k}}{\text{minimize}} \quad m \underbrace{\left( \sum_i^{n_c} -\mathbf{c}_i^\mathsf{T}\mathbf{z}_i + \sum_i^{n_c} \sum_{j \neq i} \mathbf{z}_i^\mathsf{T} P' \mathbf{z}_j \right)}_{:= \hat{g}(\mathbf{z}_{1,\dots,n_c};\boldsymbol{\theta})},
\end{aligned}
$$

where $P' = mP$. Therefore, we formulate the following optimization problem below whose objective upper bounds the original objective in Equation (3) over all feasible hash codes $\{\mathbf{h}_i\}_{i=1}^n$.

$$
\begin{aligned}
\underset{\mathbf{z}_1,\dots,\mathbf{z}_{n_c}}{\text{minimize}} \quad & \sum_i^{n_c} -\mathbf{c}_i^\mathsf{T}\mathbf{z}_i + \sum_{i,j \neq i} \mathbf{z}_i^\mathsf{T} P \mathbf{z}_j \\
\text{subject to} \quad & \mathbf{z}_i \in \{0,1\}^d, ||\mathbf{z}_i||_1 = k, \; \forall i \quad (5)
\end{aligned}
$$

In the upper bound problem above, we consider the case where the pairwise cost matrix $P$ is a diagonal matrix of non-negative values [2]. Theorem 1 in the next subsection proves that finding the optimal solution of Equation (5) is equivalent to finding the minimum cost flow solution of the flow network $G'$ illustrated in Figure 2 which can be solved efficiently and exactly in polynomial time. In practice, we use the efficient implementations from OR-Tools (Google Optimization Tools for combinatorial optimization problems) (OR-tools, 2018) to solve the minimum cost flow problem per each mini-batch.

### 4.3. Equivalence of problem 5 to minimum cost flow

**Theorem 1.** *The optimization problem in Equation (5) can be solved by finding the minimum cost flow solution on the flow network G'.*

*Proof.* Suppose we construct a complete bipartite graph $G = (A \cup B, E)$ and create a directed graph $G' = (A \cup B \cup \{s,t\}, E')$ from $G$ by adding source $s$ and sink $t$ and directing all edges $E$ in $G$ from $A$ to $B$. We also add edges from $s$ to each vertex $a_p \in A$. For each vertex $b_q \in B$, we add $n_c$ number of edges to $t$. Edges incident to $s$ have capacity $u(s, a_p) = k$ and cost $v(s, a_p) = 0$. The edges between $a_p \in A$ and $b_q \in B$ have capacity $u(a_p, b_q) = 1$ and cost $v(a_p, b_q) = -\mathbf{c}_p[q]$. Each edge $r \in \{0,\dots,n_c - 1\}$ from $b_q \in B$ to $t$ has capacity $u((b_q,t)_r) = 1$ and cost $v((b_q,t)_r) = 2\lambda_q r$. Figure 2 illustrates the flow network $G'$. The amount of flow to be sent from $s$ to $t$ is $n_c k$.

Then we define the flow $\{f_z(e)\}_{e \in E'}$, indexed both by (1) a given configuration of $\mathbf{z}_1,\dots,\mathbf{z}_{n_c}$ where each $\mathbf{z}_i \in$

---

[2]Note that we absorb the scaler factor $m$ from the definition of $P'$ and redefine $P = \mathrm{diag}(\lambda_1,\dots,\lambda_d)$.

$\{0,1\}^d, ||\mathbf{z}_i||_1 = k, \forall i$, and by (2) the edges of $G'$, below:

$$
\begin{aligned}
& (i) \; f_z(s, a_p) = k, \;\; (ii) \; f_z(a_p, b_q) = \mathbf{z}_p[q], \\
& (iii) \; f_z((b_q, t)_r) = \begin{cases} 1 & \text{for } r < \sum_{p=1}^{n_c} \mathbf{z}_p[q] \\ 0 & \text{otherwise} \end{cases} \quad (6)
\end{aligned}
$$

We first show the flow $f_z$ defined above is feasible for $G'$. The capacity constraints are satisfied by construction in Equation (6), so we only need to check the flow conservation conditions. First, the amount of input flow at $s$ is $n_c k$ and the output flow from $s$ is $\sum_{a_p \in A} f_z(s, a_p) = \sum_{a_p \in A} k = n_c k$ which is equal. The amount of input flow to each vertex $a_p \in A$ is given as $k$ and the output flow is $\sum_{b_q \in B} f_z(a_p, b_q) = \sum_q^d \mathbf{z}_p[q] = ||\mathbf{z}_p||_1 = k$.

Let us denote the amount of input flow at a vertex $b_q \in B$ as $y_q = \sum_p^{n_c} \mathbf{z}_p[q]$. The output flow from the vertex $b_q$ is $\sum_{r=0}^{n_c-1} f_z((b_q, t)_r) = \sum_{r=0}^{y_q-1} f_z((b_q, t)_r) + \sum_{r=y_q}^{n_c-1} f_z((b_q, t)_r) = y_q$ from Equation (6) $(iii)$. The last condition to check is that the amount of input flow at $t$ is equal to the output flow at $s$. $\sum_{b_q \in B} \sum_{r=0}^{n_c-1} f_z((b_q, t)_r) = \sum_{q=1}^d y_q = \sum_{q,p} \mathbf{z}_p[q] = n_c k$. This shows the construction of the flow $\{f_z(e)\}_{e \in E'}$ in Equation (6) is valid in $G'$.

Now denote $\{f_o(e)\}_{e \in E'}$ as the minimum cost flow solution of the flow network $G'$ which minimizes the total cost $\sum_{e \in E'} v(e) f_o(e)$. Denote the optimal flow from a vertex $a_p \in A$ to a vertex $b_q \in B$ as $\mathbf{z}_p'[q] := f_o(a_p, b_q)$. By the optimality of the flow $\{f_o(e)\}_{e \in E'}$, we have that $\sum_{e \in E'} v(e) f_o(e) \leq \sum_{e \in E'} v(e) f_z(e)$. By Lemma 1, the *lhs* of the inequality is equal to $\sum_p -\mathbf{c}_p^\mathsf{T}\mathbf{z}_p' + \sum_{p_1 \neq p_2} \mathbf{z}_{p_1}'^\mathsf{T} P \mathbf{z}_{p_2}'$. Also, by Lemma 2, the *rhs* is equal to $\sum_p -\mathbf{c}_p^\mathsf{T}\mathbf{z}_p + \sum_{p_1 \neq p_2} \mathbf{z}_{p_1}^\mathsf{T} P \mathbf{z}_{p_2}$.

Finally, we have that $\sum_p -\mathbf{c}_p^\mathsf{T}\mathbf{z}_p' + \sum_{p_1 \neq p_2} \mathbf{z}_{p_1}'^\mathsf{T} P \mathbf{z}_{p_2}' \leq \sum_p -\mathbf{c}_p^\mathsf{T}\mathbf{z}_p + \sum_{p_1 \neq p_2} \mathbf{z}_{p_1}^\mathsf{T} P \mathbf{z}_{p_2}, \forall \{\mathbf{z}_p\}$. Thus, we have proved that finding the minimum cost flow solution on the flow network $G'$ and translating the flows between each vertices between $A$ and $B$ as $\{\mathbf{z}_p'\}$, we can find the optimal solution to the optimization problem in Equation (5). $\square$

**Lemma 1.** *For the minimum cost flow $\{f_o(e)\}_{e \in E'}$ of the network $G'$, we have that the total cost is $\sum_{e \in E'} v(e) f_o(e) = \sum_p -\mathbf{c}_p^\mathsf{T}\mathbf{z}_p' + \sum_{p_1 \neq p_2} \mathbf{z}_{p_1}'^\mathsf{T} P \mathbf{z}_{p_2}'$.*

*Proof.* The total minimum cost $\sum_{e \in E'} v(e) f_o(e)$ is broken down as

$$\sum_{e \in E'} v(e)f_o(e) = \underbrace{\sum_{a_p \in A} v(s, a_p)f_o(s, a_p)}_{\text{flow from } s \text{ to } A} +$$

$$\underbrace{\sum_{a_p \in A} \sum_{b_q \in B} v(a_p, b_q)f_o(a_p, b_q)}_{\text{flow from } A \text{ to } B} + \underbrace{\sum_{b_q \in B} \sum_{r=0}^{n_c-1} v((b_q, t)_r)f_o((b_q, t)_r)}_{\text{flow from } B \text{ to } t}$$

Denote the amount of input flow at a vertex $b_q$ given the minimum cost flow $\{f_o(e)\}_{e \in E'}$ as $y_q' = \sum_p f_o(a_p, b_q) = \sum_p^{n_c} \mathbf{z}_p'[q]$. From the cost definition at the edges between $b_q$ and $t$, $v((b_q, t)_r) = 2\lambda_q r$, and by the optimality of the minimum cost flow, we have that $f_o((b_q, t)_r) = 1 \,\forall r < y_q'$ and $f_o((b_q, t)_r) = 0 \,\forall r \geq y_q'$. Thus, the total cost is

$$\sum_{e \in E'} v(e)f_o(e) = 0 + \sum_p^{n_c} \sum_q^d -\mathbf{c}_p[q]\mathbf{z}_p'[q] + \sum_{b_q \in B} \sum_{r=0}^{y_q'-1} 2\lambda_q r$$

$$= \sum_p -\mathbf{c}_p^\mathsf{T}\mathbf{z}_p' + \sum_q \lambda_q y_q'(y_q' - 1)$$

$$= \sum_p -\mathbf{c}_p^\mathsf{T}\mathbf{z}_p' + \sum_q \lambda_q y_q'^2 - \sum_p \sum_q \lambda_q \mathbf{z}_p'[q]$$

$$= \sum_p -\mathbf{c}_p^\mathsf{T}\mathbf{z}_p' + \sum_p \mathbf{z}_p'^\mathsf{T} P \sum_p \mathbf{z}_p' - \sum_p \mathbf{z}_p'^\mathsf{T} P \mathbf{z}_p'$$

$$= \sum_p -\mathbf{c}_p^\mathsf{T}\mathbf{z}_p' + \sum_{p_1 \neq p_2} \mathbf{z}_{p_1}'^\mathsf{T} P \mathbf{z}_{p_2}' \quad (7)$$

$\square$

**Lemma 2.** *For the $\{f_z(e)\}_{e \in E'}$ defined as Equation (6) of the network $G'$, we have that the total cost is $\sum_{e \in E'} v(e)f_z(e) = \sum_p -\mathbf{c}_p^\mathsf{T}\mathbf{z}_p + \sum_{p_1 \neq p_2} \mathbf{z}_{p_1}^\mathsf{T} P \mathbf{z}_{p_2}$.*

*Proof.* The proof is similar to Lemma 1 except that we use the definition of the flow $\{f_z(e)\}_{e \in E'}$ in Equation (6) (iii) to reduce the cost of the flow from $B$ to $t$ to $\sum_{r=0}^{y_q'-1} 2\lambda_q r$.

$\square$

**Time complexity**   For $\lambda_q \ll n_c$, note that the worst case time complexity of finding the minimum cost flow (MCF) solution in the network $G'$ is $\mathcal{O}\left((n_c + d)^2 n_c d \log(n_c + d)\right)$ (Goldberg & Tarjan, 1990). In practice, however, it has been shown that implementation heuristics such as *price updates*, *price refinement*, *push-look-ahead*, (Goldberg, 1997) and *set-relabel* (Bünnagel et al., 1998) methods drastically improve the real-life performance. Also, we emphasize again that we solve the minimum cost flow problem only *within the mini-batch* not on the entire dataset. We benchmarked the wall clock running time of the method at varying sizes of $n_c$ and $d$ and observed approximately linear time complexity in $n_c$ and $d$. Figure 1 shows the benchmark wall clock run time results.
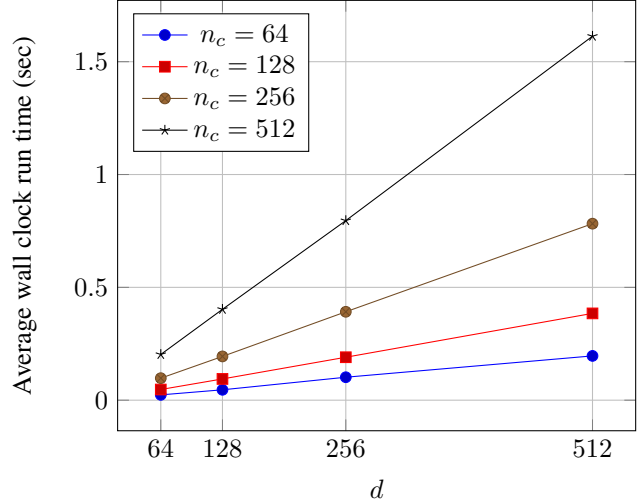


*Figure 1.* Average wall clock run time of computing minimum cost flow on $G'$ per mini-batch using (OR-tools, 2018). In practice, the run time is approximately linear in $n_c$ and $d$. Each data point is averaged over 20 runs on machines with Intel Xeon E5-2650 CPU.
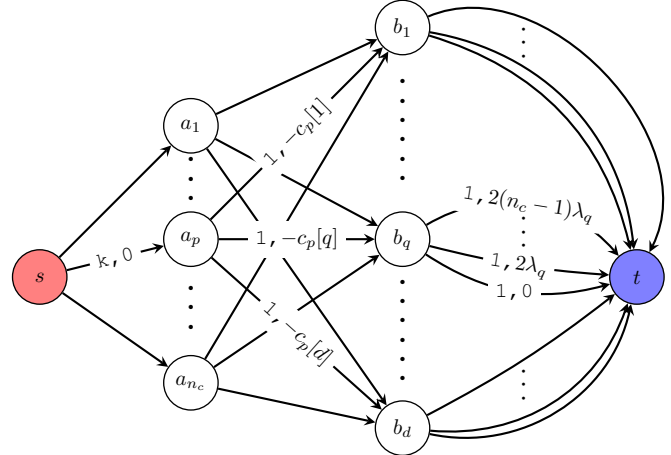


*Figure 2.* Equivalent flow network diagram $G'$ for the optimization problem in Equation (5). Labeled edges show the capacity and the cost respectively. The amount of total flow to be sent is $n_c k$.

## 4.4. Learning the embedding

As the hash codes become more and more sparse, it becomes increasingly likely for hamming distances defined on binary codes (Norouzi et al., 2012; Zhao et al., 2015) to become zero regardless of whether the input pair of data is similar or dissimilar. This phenomenon can be problematic when trained in a deep network because the back-propagation gradient would become zero and thus the embedding representations would not be updated at all. In this regard, we propose a distance function based on gated residual as shown in Equation (8). This parameterization outputs zero distance only if the embedding representations of the two input data are identical at all the hash code activations. Concretely, given a pair of embedding vectors $f(\mathbf{x}_i; \boldsymbol{\theta}), f(\mathbf{x}_j; \boldsymbol{\theta})$ and the corresponding binary k-sparse hash codes $\mathbf{h}_i, \mathbf{h}_j,$

we define the following distance function $d_{ij}^{\text{hash}}$ between the embedding vectors

$$d_{ij}^{\text{hash}} = || \, (\mathbf{h}_i \vee \mathbf{h}_j) \odot (f(\mathbf{x}_i; \boldsymbol{\theta}) - f(\mathbf{x}_j; \boldsymbol{\theta})) \, ||_1, \quad (8)$$

where $\vee$ denotes the logical *or* operation of the two binary hash codes and $\odot$ denotes the element-wise multiplication. Then, using the distance function above, we can define the following subproblems using any existing deep metric learning methods (Weinberger et al., 2006; Schroff et al., 2015; Song et al., 2016; Sohn, 2016; Song et al., 2017). Given a set of binary hash codes $\{\mathbf{h}_i\}_{i=1}^n$, we seek to solve the following subproblems where the task is to optimize the embedding representations so that similar pairs of data get hashed to the same buckets and dissimilar pairs of data get hashed to different buckets. In other words, we need similar pairs of data to have similar embedding representations indexed at the activated hash code dimensions and vice versa. In terms of the hash code optimization in Equation (4), updating the network weight has the effect of tightening the bound gap $M(\boldsymbol{\theta})$.

Equation (9) and Equation (10) show the subproblems defined on the distance function above using *Triplet* (Schroff et al., 2015) and *Npairs* (Sohn, 2016) method respectively. We optimize these embedding subproblems by updating the network parameter $\boldsymbol{\theta}$ via stochastic gradient descent using the subgradients $\frac{\partial \ell_{\text{metric}}(\boldsymbol{\theta}; \mathbf{h}_{1,\dots,n})}{\partial \boldsymbol{\theta}}$ given the hash codes per each mini-batch.

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \underbrace{\frac{1}{|\mathcal{T}|} \sum_{(i,j,k) \in \mathcal{T}} [d_{ij}^{\text{hash}} + \alpha - d_{ik}^{\text{hash}}]_+}_{\ell_{\text{triplet}}(\boldsymbol{\theta}; \, \mathbf{h}_{1,\dots,n})}$$

$$\text{subject to} \quad ||f(\mathbf{x}; \boldsymbol{\theta})||_2 = 1, \quad (9)$$

where $\mathcal{T} = \{(\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-)\}_i$ is the set of triplets (Schroff et al., 2015), and the embedding vectors are normalized onto unit hypersphere $||f(\mathbf{x}; \boldsymbol{\theta})||_2 = 1, \, \forall \mathbf{x} \in \mathcal{X}$. We also apply the *semi-hard negative mining* procedure (Schroff et al., 2015) where hard negatives farther than the distance between the anchor and positives are mined within the mini-batch. In practice, since our method can be applied to any deep metric learning methods, we use existing deep metric learning implementations available in `tf.contrib.losses.metric_learning`. Similarly, we could also employ *npairs* (Sohn, 2016) method,

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \underbrace{\frac{-1}{|\mathcal{P}|} \sum_{(i,j) \in \mathcal{P}} \log \frac{\exp(-d_{ij}^{\text{hash}})}{\exp(-d_{ij}^{\text{hash}}) + \sum_{k:y_k \neq y_i} \exp(-d_{ik}^{\text{hash}})}}_{\ell_{\text{npairs}}(\boldsymbol{\theta}; \, \mathbf{h}_{1,\dots,n})}$$

$$+ \frac{\lambda}{m} \sum_i ||f(\mathbf{x}_i; \boldsymbol{\theta})||_2^2, \quad (10)$$

where the *npairs* mini-batch $B$ is constructed with positive pairs $(\mathbf{x}, \mathbf{x}^+)$ which are negative with respect to all

other pairs. $B = \{(\mathbf{x}_1, \mathbf{x}_1^+), \dots, (\mathbf{x}_n, \mathbf{x}_n^+))\}$ and $\mathcal{P}$ denotes the set of all positive pairs within the mini-batch. We use the existing implementation of npairs loss in Tensorflow as well. Note that even though the distance $d_{ij}^{\text{hash}}$ is defined *after* masking the embeddings with the union binary vector $(\mathbf{h}_i \vee \mathbf{h}_j)$, it's important to normalize or regularize the embedding representation *before* the masking operations for the optimization stability due to the sparse nature of the hash codes.

---

**Algorithm 1** Learning algorithm

---

**input** $\boldsymbol{\theta}_b^{\text{emb}}$ (pretrained metric learning base model); $\boldsymbol{\theta}_d \in \mathbb{R}^d$
**initialize** $\boldsymbol{\theta}_f = [\boldsymbol{\theta}_b, \boldsymbol{\theta}_d]$
1: **for** $t = 1, \dots, \text{MAXITER}$ **do**
2:     Sample a minibatch $\{\mathbf{x}_j\}$
3:     Update the flow network $G'$ by recomputing the cost vectors for all classes in the minibatch
    $\mathbf{c}_i = \frac{1}{m} \sum_{k:y_k=i} f(\mathbf{x}_k; \boldsymbol{\theta}_f)$
4:     Compute the hash codes $\{\mathbf{h}_i\}$ minimizing Equation (5) via finding the minimum cost flow on $G'$
5:     Update the network parameter given the hash codes
    $\boldsymbol{\theta}_f \leftarrow \boldsymbol{\theta}_f - \eta^{(t)} \partial \ell_{\text{metric}}(\boldsymbol{\theta}_f; \, \mathbf{h}_{1,\dots,n_c}) / \partial \boldsymbol{\theta}_f$
6:     Update stepsize $\eta^{(t)} \leftarrow$ ADAM rule (Kingma & Ba, 2014)
7: **end for**
**output** $\boldsymbol{\theta}_f$ (final estimate);

---

### 4.5. Query efficiency analysis

In this subsection, we examine the expectation and the variance of the query time speed up over linear search. Recall the properties of the compound hash code defined in Section 3, $\mathbf{h} \in \{0, 1\}^d$ and $||\mathbf{h}||_1 = k$. Given $n$ such hash codes, we have that $\Pr(\mathbf{h}_i^\intercal \mathbf{h}_j = 0) = \binom{d-k}{k} / \binom{d}{k}$ assuming the hash code uniformly distributes the items throughout different buckets. For a given hash code $\mathbf{h}_q$, the number of retrieved data is $N_q = \sum_{i \neq q} \mathbb{1}(\mathbf{h}_i^\intercal \mathbf{h}_q \neq 0)$. Then, the expected number of retrieved data is $\mathbb{E}[N_q] = (n-1)\left(1 - \binom{d-k}{k} / \binom{d}{k}\right)$. Thus, in contrast to linear search, the expected speedup factor (SUF) under perfectly uniform distribution of the hash code is

$$\mathbb{E}[\text{SUF}] = \left(1 - \frac{\binom{d-k}{k}}{\binom{d}{k}}\right)^{-1} \quad (11)$$

In the case where $d \gg k$, the speedup factor approaches $\left(\frac{d}{k^2}\right)$. Similarly, we have that the variance is $V[N_q] = (n-1)\left(1 - \binom{d-k}{k} / \binom{d}{k}\right)\binom{d-k}{k} / \binom{d}{k}$.

## 5. Implementation details

**Network architecture** In our experiments, we used the NIN (Lin et al., 2013) architecture (denote the parameters as $\boldsymbol{\theta}_b$) with *leaky relu* (Xu et al., 2015) with $\alpha = 5.5$ as activation function and trained Triplet embedding network with semi-hard negative mining (Schroff et al., 2015) and

Npairs network (Sohn, 2016) from scratch as the base model. We snapshot the network weights ($\theta_b^{\text{emb}}$) of the learned base model. Then we replace the last layer in ($\theta_b^{\text{emb}}$) with a randomly initialized $d$ dimensional fully connected projection layer ($\theta_d$) and finetune the hash network (denote the parameters as $\theta_f = [\theta_b, \theta_d]$). Algorithm 1 summarizes the training procedure in detail.

**Hash table construction and query** We use the learned hash network $\theta_f$ and apply Equation (1) to convert a hash data $\mathbf{x}_i$ into the hash code $\mathbf{h}(\mathbf{x}_i; \theta_f)$ and use the base embedding network $\theta_b^{\text{emb}}$ to convert the data into the embedding representation $f(\mathbf{x}_i; \theta_b^{\text{emb}})$. Then, the embedding representation is hashed to buckets corresponding to the $k$ set bits in the hash code. We use the similar procedure and convert a query data $\mathbf{x}_q$ into the hash code $\mathbf{h}(\mathbf{x}_q; \theta_f)$ and into the embedding representation $f(\mathbf{x}_q; \theta_b^{\text{emb}})$. Once we retrieve the union of all bucket items indexed at the $k$ set bits in the hash code, we apply a reranking procedure (Wang et al., 2016) based on the euclidean distance in the embedding representation space.

**Evaluation metrics** We report our accuracy results using precision@k (Pr@k) and normalized mutual information (NMI) (Manning et al., 2008) metrics. Precision@k is computed based on the reranked ordering (described above) of the retrieved items from the hash table. We evaluate NMI, when the code sparsity is set to $k = 1$, treating each bucket as individual clusters. In this setting, NMI becomes perfect, if each bucket has perfect class purity (pathologically putting one item per each bucket is prevented by construction since $d \ll n$). We report the speedup results by comparing the number of retrieved items versus the total number of data (exhaustive linear search) and denote this metric as SUF. As the hash code becomes uniformly distributed, SUF metric approaches the theoretical expected speedup in Equation (11). Figure 3 shows that the measured SUF of our method closely follows the theoretical upper bound in contrast to other methods.

# 6. Experiments

We report our results on Cifar-100 (Krizhevsky et al., 2009) and ImageNet (Russakovsky et al., 2015) datasets and compare the accuracy against several baseline methods. First baseline methods are the state of the art deep metric learning models (Schroff et al., 2015; Sohn, 2016) performing an exhaustive linear search over the whole dataset given a query data. Another baselines are the Binarization transform (Agrawal et al., 2014; Zhai et al., 2017) methods where the dimensions of the hash code corresponding to the top $k$ dimensions of the embedding representation are set. We also perform vector quantization (Wang et al., 2016) on the learned embedding representation from the deep metric learning methods above on the entire dataset and compute the hash code based on the indices of the $k$ nearest centroids.

'Triplet' and 'Npairs' denotes the deep metric learning base models performing an exhaust linear search per each query. 'Th' denotes the binarization transform baseline, 'VQ' denotes the vector quantization baseline.

## 6.1. Cifar-100

Cifar-100 (Krizhevsky et al., 2009) dataset has 100 classes. Each class has 500 images for *train* and 100 images for *test*. Given a query image from *test*, we experiment the search performance both when the hash table is constructed from *train* and from *test*. We subtract the per-pixel mean of training images across all the images and augmented the dataset by zero-padding 4 pixels on each side, randomly cropping $32 \times 32$, and applying random horizontal flipping. The batch size is set to 128. The metric learning base model is trained for 175k iterations, and learning rate decays to 0.1 of previous learning rate after 100k iterations. We finetune the base model for 70k iterations and decayed the learning rate to 0.1 of previous learning rate after 40k iterations. Table 1 show results using the triplet network with $d = 256$ and Table 2 show results using the npairs network with $d = 64$. The results show that our method not only outperforms search accuracies of the state of the art deep metric learning base models but also provides up to $98\times$ speed up over exhaustive search.

| | | | *train* | | | | *test* | | |
|---|---|---|---|---|---|---|---|---|---|
| | Method | SUF | Pr@1 | Pr@4 | Pr@16 | SUF | Pr@1 | Pr@4 | Pr@16 |
| | Triplet | 1.00 | 62.64 | 61.91 | 61.22 | 1.00 | 56.78 | 55.99 | 53.95 |
| $k=1$ | Triplet-Th | 43.19 | 61.56 | 60.24 | 58.23 | 41.21 | 54.82 | 52.88 | 48.03 |
| | Triplet-VQ | 40.35 | 62.54 | 61.78 | 60.98 | 22.78 | 56.74 | 55.94 | 53.77 |
| | Triplet-Ours | **97.77** | **63.85** | **63.40** | **63.39** | **97.67** | **57.63** | **57.16** | **55.76** |
| $k=2$ | Triplet-Th | 15.34 | 62.41 | 61.68 | 60.89 | 14.82 | 56.55 | 55.62 | 52.90 |
| | Triplet-VQ | 6.94 | 62.66 | 61.92 | 61.26 | 5.63 | 56.78 | 56.00 | 53.99 |
| | Triplet-Ours | **78.28** | **63.60** | **63.19** | **63.09** | **76.12** | **57.30** | **56.70** | **55.19** |
| $k=3$ | Triplet-Th | 8.04 | 62.66 | 61.88 | 61.16 | 7.84 | 56.78 | 55.91 | 53.64 |
| | Triplet-VQ | 2.96 | 62.62 | 61.92 | 61.22 | 2.83 | 56.78 | 55.99 | 53.95 |
| | Triplet-Ours | **44.36** | **62.87** | **62.22** | **61.84** | **42.12** | **56.97** | **56.25** | **54.40** |
| $k=4$ | Triplet-Th | 5.00 | 62.66 | 61.94 | 61.24 | 4.90 | 56.84 | 56.01 | 53.86 |
| | Triplet-VQ | 1.97 | 62.62 | 61.91 | 61.22 | 1.91 | 56.77 | 55.99 | 53.94 |
| | Triplet-Ours | **16.52** | **62.81** | **62.14** | **61.58** | **16.19** | **57.11** | **56.21** | **54.20** |

*Table 1.* Results with Triplet network with hard negative mining. Querying Cifar-100 *test* data against hash tables built on *train* set and on *test* set.

| | | | *train* | | | | *test* | | |
|---|---|---|---|---|---|---|---|---|---|
| | Method | SUF | Pr@1 | Pr@4 | Pr@16 | SUF | Pr@1 | Pr@4 | Pr@16 |
| | Npairs | 1.00 | 61.78 | 60.63 | 59.73 | 1.00 | 57.05 | 55.70 | 53.91 |
| $k=1$ | Npairs-Th | 13.65 | 60.80 | 59.49 | 57.27 | 12.72 | 54.95 | 52.60 | 47.16 |
| | Npairs-VQ | 31.35 | 61.22 | 60.24 | 59.34 | 34.86 | 56.76 | 55.35 | 53.75 |
| | Npairs-Ours | **54.90** | **63.11** | **62.29** | **61.94** | **54.85** | **58.19** | **57.22** | **55.87** |
| $k=2$ | Npairs-Th | 5.36 | 61.65 | 60.50 | 59.50 | 5.09 | 56.52 | 55.28 | 53.04 |
| | Npairs-VQ | 5.44 | 61.82 | 60.56 | 59.70 | 6.08 | 57.13 | 55.74 | 53.90 |
| | Npairs-Ours | **16.51** | **61.98** | **60.93** | **60.15** | **16.20** | **57.27** | **55.98** | **54.42** |
| $k=3$ | Npairs-Th | 3.21 | 61.75 | 60.66 | 59.73 | 3.10 | 56.97 | 55.56 | 53.76 |
| | Npairs-VQ | 2.36 | 61.78 | 60.62 | 59.73 | 2.66 | 57.01 | 55.69 | 53.90 |
| | Npairs-Ours | **7.32** | **61.90** | **60.80** | **59.96** | **7.25** | **57.15** | **55.81** | **54.10** |
| $k=4$ | Npairs-Th | 2.30 | 61.78 | 60.66 | 59.75 | 2.25 | 57.02 | 55.64 | 53.88 |
| | Npairs-VQ | 1.55 | 61.78 | 60.62 | 59.73 | 1.66 | 57.03 | 55.70 | 53.91 |
| | Npairs-Ours | **4.52** | **61.81** | **60.69** | **59.77** | **4.51** | **57.15** | **55.77** | **54.01** |

*Table 2.* Results with Npairs (Sohn, 2016) network. Querying Cifar-100 *test* data against hash tables built on *train* set and on *test* set.

## 6.2. ImageNet

ImageNet ILSVRC-2012 (Russakovsky et al., 2015) dataset has $1,000$ classes and comes with *train* ($1,281,167$ images) and *val* set ($50,000$ images). We use the first nine splits of *train* set to train our model, the last split of *train* set for validation, and use *validation* dataset to test the query performance. We use the images downsampled to $32 \times 32$ from (Chrabaszcz et al., 2017). Preprocessing step is identical with cifar-100 and we used the pixel mean provided in the dataset. The batch size for the metric learning base model is set to $512$ and is trained for 450k iterations, and learning rate decays to $0.3$ of previous learning rate after each 200k iterations. When we finetune npairs base model for $d=512$, we set the batch size to 1024 and total iterations to 35k with decaying the learning rate to $0.3$ of previous learning rate after each 15k iterations. When we finetune the triplet base model for $d=256$, we set the batch size to 512 and total iterations to 70k with decaying the learning rate to $0.3$ of previous learning rate after each 30k iterations. Our results in Table 3 and Table 4 show that our method outperforms the state of the art deep metric learning base models in search accuracy while providing up to $478\times$ speed up over exhaustive linear search. Table 5 compares the NMI metric and shows that the hash table constructed from our representation yields buckets with significantly better class purity on both datasets and on both methods.

| | Method | SUF | Pr@1 | Pr@4 | Pr@16 |
|---|---|---|---|---|---|
| | Npairs | 1.00 | 15.73 | 13.75 | 11.08 |
| $k=1$ | Npairs-Th | 1.74 | 15.06 | 12.92 | 9.92 |
| | Npairs-VQ | 451.42 | 15.20 | 13.27 | 10.96 |
| | Npairs-Ours | **478.46** | **16.95** | **15.27** | **13.06** |
| $k=2$ | Npairs-Th | 1.18 | 15.70 | 13.69 | 10.96 |
| | Npairs-VQ | 116.26 | 15.62 | 13.68 | 11.15 |
| | Npairs-Ours | **116.61** | **16.40** | **14.49** | **12.00** |
| $k=3$ | Npairs-Th | 1.07 | 15.73 | 13.74 | 11.07 |
| | Npairs-VQ | **55.80** | 15.74 | 13.74 | 11.12 |
| | Npairs-Ours | 53.98 | **16.24** | **14.32** | **11.73** |

*Table 3.* Results with Npairs (Sohn, 2016) network. Querying ImageNet *val* data against hash table built on *val* set.



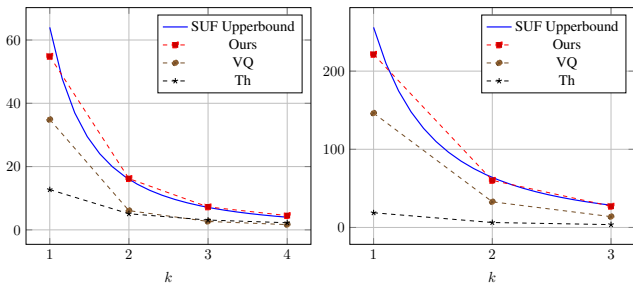*Figure 3.* SUF metric for on Cifar-100 and ImageNet respectively.

| | Method | SUF | Pr@1 | Pr@4 | Pr@16 |
|---|---|---|---|---|---|
| | Triplet | 1.00 | 10.90 | 9.39 | 7.45 |
| $k=1$ | Triplet-Th | 18.81 | 10.20 | 8.58 | 6.50 |
| | Triplet-VQ | 146.26 | 10.37 | 8.84 | 6.90 |
| | Triplet-Ours | **221.49** | **11.00** | **9.59** | **7.83** |
| $k=2$ | Triplet-Th | 6.33 | 10.82 | 9.30 | 7.32 |
| | Triplet-VQ | 32.83 | 10.88 | 9.33 | 7.39 |
| | Triplet-Ours | **60.25** | **11.10** | **9.64** | **7.73** |
| $k=3$ | Triplet-Th | 3.64 | 10.87 | 9.38 | 7.42 |
| | Triplet-VQ | 13.85 | 10.90 | 9.38 | 7.44 |
| | Triplet-Ours | **27.16** | **11.20** | **9.55** | **7.60** |

*Table 4.* Results with Triplet network with hard negative mining. Querying ImageNet *val* data against hash table built on *val* set.

| | Cifar-100 | | ImageNet |
|---|---|---|---|
| | train | test | val |
| Triplet-Th | 68.20 | 54.95 | 31.62 |
| Triplet-VQ | 76.85 | 62.68 | 45.47 |
| Triplet-Ours | **89.11** | **68.95** | **48.52** |
| Npairs-Th | 51.46 | 44.32 | 15.20 |
| Npairs-VQ | 80.25 | 66.69 | 53.74 |
| Npairs-Ours | **84.90** | **68.56** | **55.09** |

*Table 5.* Hash table NMI for Cifar-100 and Imagenet.

## 7. Conclusion

We have presented a novel end-to-end optimization algorithm for jointly learning a quantizable embedding representation and the sparse binary hash code which then can be used to construct a hash table for efficient inference. We also show an interesting connection between finding the optimal sparse binary hash code and solving a *minimum cost flow* problem. Our experiments show that the proposed algorithm not only achieves the state of the art search accuracy outperforming the previous state of the art deep metric learning approaches (Schroff et al., 2015; Sohn, 2016) but also provides up to $98\times$ and $478\times$ search speedup on Cifar-100 and ImageNet datasets respectively.

## Acknowledgements

# References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

Agrawal, P., Girshick, R., and Malik, J. Analyzing the performance of multilayer neural networks for object recognition. In *ECCV*, 2014.

Bell, S. and Bala, K. Learning visual similarity for product design with convolutional neural networks. In *SIGGRAPH*, 2015.

Boykov, Y., Veksler, O., and Zabih, R. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 2001.

Bromley, J., Guyon, I., Lecun, Y., Sackinger, E., and Shah, R. Signature verification using a "siamese" time delay neural network. In *NIPS*, 1994.

Bünnagel, U., Korte, B., and Vygen, J. Efficient implementation of the goldberg–tarjan minimum-cost flow algorithm. *Optimization Methods and Software*, 10(2): 157–174, 1998.

Cao, Y., Long, M., Wang, J., Zhu, H., and Wen, Q. Deep quantization network for efficient image retrieval. In *AAAI*, 2016.

Choromanska, A., Agarwal, A., and Langford, J. Extreme multi class classification. In *NIPS*, 2013.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Goldberg, A. V. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of algorithms*, 22 (1):1–29, 1997.

Goldberg, A. V. and Tarjan, R. E. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3), 1990.

Hadsell, R., Chopra, S., and Lecun, Y. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-100 (canadian institute for advanced research). 2009. URL http://www.cs.toronto.edu/~kriz/cifar.html.

Li, Q., Sun, Z., He, R., and Tan, T. Deep supervised discrete hashing. In *NIPS*, 2017.

Lin, M., Chen, Q., and Yan, S. Network in network. *CoRR*, abs/1312.4400, 2013.

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

Liu, S. and Lu, H. Learning deep representations with diode loss for quantization-based similarity search. In *IJCNN*, 2017.

Manning, C. D., Raghavan, P., and Schutze, H. *Introduction to Information Retrieval*. Cambridge university press, 2008.

Norouzi, M., Fleet, D. J., and Salakhutdinov, R. R. Hamming distance metric learning. In *NIPS*, 2012.

OR-tools, G. https://developers.google.com/optimization/, 2018.

Prabhu, Y. and Varma, M. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *SIGKDD*, 2014.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

Schroff, F., Kalenichenko, D., and Philbin, J. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015.

Sener, O., Song, H. O., Saxena, A., and Savarese, S. Learning transferrable representations for unsupervised domain adaptation. In *NIPS*, 2016.

Shen, F., Shen, C., Liu, W., and Shen, H. T. Supervised discrete hashing. In *CVPR*, 2015.

Sohn, K. Improved deep metric learning with multi-class n-pair loss objective. In *NIPS*, 2016.

Song, H. O., Xiang, Y., Jegelka, S., and Savarese, S. Deep metric learning via lifted structured feature embedding. In *CVPR*, 2016.

Song, H. O., Jegelka, S., Rathod, V., and Murphy, K. Deep metric learning via facility location. In *CVPR*, 2017.

Wang, J., Zhang, T., Song, J., Sebe, N., and Shen, H. T. A survey on learning to hash. *arXiv preprint arXiv:1606.00185*, 2016.

Wang, X. and Gupta, A. Unsupervised learning of visual representations using videos. In *ICCV*, 2015.

Weinberger, K. Q., Blitzer, J., and Saul, L. K. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, 2006.

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

Xu, B., Wang, N., Chen, T., and Li, M. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Zhai, A., Kislyuk, D., Jing, Y., Feng, M., Tzeng, E., Donahue, J., Du, Y. L., and Darrell, T. Visual discovery at pinterest. In *Proceedings of the 26th International Conference on World Wide Web Companion*, 2017.

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