End-to-End Efficient Representation Learning via Cascading Combinatorial Optimization

Yeonwoo Jeong, Yoonsung Kim, Hyun Oh Song
Department of Computer Science and Engineering, Seoul National University, Seoul, Korea
{yeonwoo, yskim227, hyunoh}@mllab.snu.ac.kr

Abstract

We develop hierarchically quantized efficient embedding representations for similarity-based search and show that this representation provides not only the state of the art performance on the search accuracy but also provides several orders of speed up during inference. The idea is to hierarchically quantize the representation so that the quantization granularity is greatly increased while maintaining the accuracy and keeping the computational complexity low. We also show that the problem of finding the optimal sparse compound hash code respecting the hierarchical structure can be optimized in polynomial time via minimum cost flow in an equivalent flow network. This allows us to train the method end-to-end in a mini-batch stochastic gradient descent setting. Our experiments on Cifar100 and ImageNet datasets show the state of the art search accuracy while providing several orders of magnitude search speedup respectively over exhaustive linear search over the dataset.

1. Introduction

Learning the feature embedding representation that preserves the notion of similarities among the data is of great practical importance in machine learning and vision and is at the basis of modern similarity-based search [21, 23], verification [26], clustering [2], retrieval [25, 24], zero-shot learning [31, 5], and other related tasks. In this regard, deep metric learning methods [2, 21, 23] have shown advances in various embedding tasks by training deep convolutional neural networks end-to-end encouraging similar pairs of data to be close to each other and dissimilar pairs to be farther apart in the embedding space.

Despite the progress in improving the embedding representation accuracy, improving the inference efficiency and scalability of the representation in an end-to-end optimization framework is relatively less studied. Practitioners deploying the method on large-scale applications often resort to employing post-processing techniques such as embedding thresholding [1, 32] and vector quantization [27] at the cost of the loss in the representation accuracy. Recently, Jeong & Song [11] proposed an end-to-end learning algorithm for quantizable representations which jointly optimizes the quality of the convolutional neural network based embedding representation and the performance of the corresponding sparsity constrained compound binary hash code and showed significant retrieval speedup on ImageNet [20] without compromising the accuracy.

In this work, we seek to learn hierarchically quantizable representations and propose a novel end-to-end learning method significantly increasing the quantization granularity while keeping the time and space complexity manageable so the method can still be efficiently trained in a mini-batch stochastic gradient descent setting. Besides the efficiency issues, however, naively increasing the quantization granularity could cause severe degradation in the search accuracy or lead to dead buckets hindering the search speedup.

To this end, our method jointly optimizes both the sparse compound hash code and the corresponding embedding representation respecting a hierarchical structure. We alternate between performing cascading optimization of the optimal sparse compound hash code per each level in the hierarchy and updating the neural network to adjust the corresponding embedding representations at the active bits of the compound hash code.

Our proposed learning method outperforms both the reported results in [11] and the state of the art deep metric learning methods [21, 23] in retrieval and clustering tasks on Cifar-100 [13] and ImageNet [20] datasets while, to the best of our knowledge, providing the highest reported inference speedup on each dataset over exhaustive linear search.

2. Related works

Embedding representation learning with neural networks has its roots in Siamese networks [4, 9] where it was trained end-to-end to pull similar examples close to each other and push dissimilar examples at least some margin away from each other in the embedding space. [4] demonstrated the idea could be used for signature verification tasks. The line of work since then has been explored in wide variety of practical applications such as face recognition [26], domain adaptation
illustrates an example translation from the given
activated maintaining the hard constraint on the sparsity of hash
functions. Although comparing binary hamming codes is
more efficient than comparing continuous embedding representa-
tions, this still requires the linear search over the entire
dataset which is not likely to be as efficient for large scale
problems. [7, 16] seek to vector quantize the dataset and
back propagate the metric loss, however, it requires repeatedly
running k-means clustering on the entire dataset during
training with prohibitive computational complexity.

We seek to jointly learn the hierarchically quantizable
embedding constrained binary hash code in an efficient mini-batch based
draw-end-to-end learning framework. Jeong & Song [11] moti-
vated maintaining the hard constraint on the sparsity of hash
code to provide guaranteed retrieval inference speedup by
only considering \( k_s \) out of \( d \) buckets and thus avoiding linear
search over the dataset. We also explicitly maintain this con-
straint, but at the same time, greatly increasing the number of
representable buckets by imposing an efficient hierarchi-
ical structure on the hash code to unlock significant improvement
in the speedup factor.

3. Problem formulation

Consider the following hash function

\[
 r(x) = \arg\min_{h \in \{0,1\}^d} f(x;\theta)\top h
\]

under the constraint that \( \|h\|_1 = k_s \). The idea is to optimize the
weights in the neural network \( f(\cdot;\theta) : \mathcal{X} \to \mathbb{R}^d \), take \( k_s \)
high activation dimensions, activate the corresponding
dimensions in the binary compound hash code \( h \), and hash
the data \( x \in \mathcal{X} \) into the corresponding active buckets of a
hash table \( \mathcal{H} \). During inference, a query \( x_q \) is given, and all
the hashed items in the \( k_s \) active bits set by the hash
function \( r(x_q) \) are retrieved as the candidate nearest items.

Often times [27], these candidates are reranked based on the
euclidean distance in the base embedding representation
\( f(\cdot;\theta) \) space.

Given a query \( h_q \), the expected number of retrieved items is
\( \sum_{x \in \mathcal{X}} \Pr(h_q\top h_q \neq 0) \). Then, the expected speedup factor
[11] (SUf) is the ratio between the total number of items and the
expected number of retrieved items. Concretely, it becomes
\( \Pr(h_q\top h_q \neq 0) \approx 1 - (\frac{(d-k_s)}{k_s})^{-1} \). In case \( d \gg k_s \), this ratio approaches \( d/k_s^2 \).

Now, suppose we design a hash function \( r(x) \) so that the
function has total \( \text{dim}(r(x)) = d^k \) (i.e., exponential in
some integer parameter \( k > 1 \)) indexable buckets. The
expected speedup factor [11] approaches \( d^k/k_s^2 \) which means
the query time speedup increases linearly with the number of
buckets. However, naively increasing the bucket size
for higher speedup has several major downsides. First, the
hashing network has to output and hold \( d^k \) activations in the
memory at the final layer which can be unpractical in terms of
the space efficiency for large scale applications. Also, this
could also lead to dead buckets which are under-utilized and
degrade the search speedup. On the other hand, hashing the
items uniformly at random among the buckets could help to
alleviate the dead buckets but this could lead to a severe
drop in the search accuracy.

Our approach to this problem of maintaining a large num-
ber of representable buckets while preserving the accuracy
and keeping the computational complexity manageable is to
enforce a hierarchy among the optimal hash codes in an effi-
cient tree structure. First, we use \( \text{dim}(f(x)) = dk \) number of
activations instead of \( d^k \) activations in the last layer of the
hash network. Then, we define the unique mapping between
the \( dk \) activations to \( d^k \) buckets by the following procedure.

Denote the hash code as \( h = [h^1, \ldots, h^k] \in \{0,1\}^{d \times k} \)
where \( \|h^v\|_1 = 1 \) \( \forall v \neq k \) and \( \|h^k\|_1 = k_s \). The superscript
denotes the level index in the hierarchy. Now, suppose we
construct a tree \( T \) with branching factor \( d \), depth \( k \) where
the root node has the level index of 0. Let each \( d^k \)
leaf node in \( T \) represent a bucket indexed by the hash function
\( r(x) \). Then, we can interpret each \( h^v \) vector to indicate the
branching from depth \( v - 1 \) to depth \( v \) in \( T \). Note, from
the construction of \( h \), the branching is unique until level
\( k - 1 \), but the last branching to the leaf nodes is multi-way
because \( k_s \) bits are set due to the sparsity constraint at level
\( k \). Figure 1 illustrates an example translation from the given
hash activation to the tree bucket index for \( k = 2 \) and \( k_s = 2 \).

Concretely, the hash function \( r(x) : \mathbb{R}^{d \times k} \to \{0,1\}^{d^k} \)
can be expressed compactly as Equation (1).

\[
 r(x) = \bigotimes_{v=1}^k \arg\min_{h^v} -(f(x;\theta)^v)\top h^v \quad (1)
\]

subject to \( \|h^v\|_1 = \begin{cases} 1 & \forall v \neq k \\ k_s & \text{ and } h^v \in \{0,1\}^d \end{cases} \)

where \( \bigotimes \) denotes the tensor multiplication operator between
two vectors. The following section discusses how to find
the optimal hash code \( h \) and the corresponding activation
\( f(x;\theta) = [f(x;\theta)^1, \ldots, f(x;\theta)^k] \in \mathbb{R}^{d \times k} \)
respecting the hierarchical structure of the code.

4. Methods

To compute the optimal set of embedding representations and
the corresponding hash code, the embedding representations
are first required in order to infer which \( k_s \) activations to
set in the hash code, but to learn the embedding representa-
tions, it requires the hash code to determine which dimensions of the activations to adjust so that similar items
would get hashed to the same buckets and vice versa. We
take the alternating minimization approach iterating over
computing the sparse hash codes respecting the hierarchical
quantization structure and updating the network parameters
indexed at the given hash codes per each mini-batch. Section 4.1 and Section 4.3 formalize the subproblems in detail.

4.1. Learning the hierarchical hash code

Given a set of continuous embedding representation \( \{ f(x_i; \theta) \}_{i=1}^n \), we wish to compute the optimal binary hash code \( \{ h_{1i}, \ldots, h_{ni} \} \) so as to hash similar items to the same buckets and dissimilar items to different buckets. Furthermore, we seek to constrain the hash code to simultaneously maintain the hierarchical structure and the hard sparsity conditions throughout the optimization process. Suppose items \( x_1 \) and \( x_2 \) are dissimilar items, in order to hash the two items to different buckets, at each level of \( \mathcal{T} \), we seek to encourage the hash code for each item at level \( v \), \( h_{1i}^v \) and \( h_{2i}^v \) to differ. To achieve this, we optimize the hash code for all items per each level sequentially in cascading fashion starting from the first level \( \{ h_{1i}^1, \ldots, h_{ni}^1 \} \) to the leaf nodes \( \{ h_{1i}^k, \ldots, h_{ni}^k \} \) as shown in Equation (2).

\[
\begin{align*}
\min_{h_{1i}^1, \ldots, h_{ni}^n} & \quad \sum_{v=1}^{k} \sum_{i=1}^{n} \left( -f(x_i; \theta)^\top h_{1i}^v \right) \\
\text{s.t.} & \quad \|h_{1i}^v\| = 1, \quad \forall v \not= k, \quad h_{1i}^v \in \{0,1\}^d, \quad \forall i, \quad (i,j) \in \mathcal{N} \\
\end{align*}
\]

subject to \( \|h_{1i}^v\| = 1, \forall v \not= k, h_{1i}^v \in \{0,1\}^d, \forall i, \) where \( \mathcal{N} \) denotes the set of dissimilar pairs of data and \( \mathbb{I}(\cdot) \) denotes the indicator function. Concretely, given the hash codes from all the previous levels, we seek to minimize the following discrete optimization problem in Equation (3), subject to the same constraints as in Equation (2), sequentially for all levels \( v \in \{1, \ldots, k\} \). The unary term in the objective encourages selecting as large embedding vector as possible while the second term loops over all pairs of dissimilar siblings and penalizes for their orthogonality. The last term encourages selecting as orthogonal elements as possible for a pair of hash codes from different classes in the current level \( v \). The last term also makes sure, in the event that the second term becomes zero, the hash code still respects orthogonality among dissimilar items. This can occur when the hash code for all the previous levels was computed perfectly splitting dissimilar pairs into different branches and the second term becomes zero.

\[
\begin{align*}
\min_{h_{11}, \ldots, h_{nn}} & \quad \sum_{v=1}^{n} \left( -f(x_i; \theta)^\top h_i^v \right) + \sum_{(i,j) \in \mathcal{N}^v} h_i^v h_j^v + \sum_{(i,j) \in \mathcal{N}} h_i^v P^v h_j^v \\
\text{s.t.} & \quad \|h_{1i}^v\| = 1, \quad \forall v \not= k, \quad h_{1i}^v \in \{0,1\}^d, \quad \forall i, \quad (i,j) \in \mathcal{N} \\
\end{align*}
\]

where \( \mathcal{N}^v = \{(i,j) \in \mathcal{N} | h_i^v = h_j^v, \forall w = 1, \ldots, v - 1 \} \) denotes the set of pairs of siblings at level \( v \) in \( \mathcal{T} \), and \( Q^v, P^v \) encodes the pairwise cost for the sibling and the orthogonality terms respectively. However, optimizing Equation (3) is NP-hard in general even in the simpler case of \( k_v = 1, k = 1, d > 2 \) [3, 11]. Inspired by [11], we use the average embedding of each class within the minibatch \( c_{i}^p = \frac{1}{m} \sum_{i:y_i=p} f(x_i; \theta)^\top \in \mathbb{R}^d \) as shown in Equation (4).

\[
\begin{align*}
\min_{z_1, \ldots, z_{m}} & \quad \sum_{p=1}^{n_c} (-c_{i}^p)^\top z_p + \sum_{(p,q) \in \mathcal{N}^p} z_p^\top Q z_q + \sum_{p \not= q} z_p^\top P z_q \\
\text{s.t.} & \quad \|z_p\| = \begin{cases} 1 & \forall v \not= k, \quad k = 1, \ldots, m, \quad z_p \in \{0,1\}^d, \quad \forall p \end{cases}, \\
\end{align*}
\]

where \( \mathcal{N}^p = \{(p,q) | z_p^w = z_q^w, \forall w = 1, \ldots, v - 1 \} \), \( n_c \) is the number of unique classes in the minibatch, and we assume each class has \( m \) examples in the minibatch (i.e. \( npairs \) [23] minibatch construction). Note, in accordance with the deep metric learning problem setting [21, 23, 11], we assume we are given access to the label adjacency information only within the minibatch.

The objective in Equation (4) upperbounds the objective in Equation (3) (denote as \( g(\cdot; \theta) \)) by a gap \( M(\theta) \) which depends only on \( \theta \). Concretely, rewriting the summation in the unary term in \( g \), we get

\[
g(h_1, \ldots, h_n; \theta) = \sum_{p=1}^{n_c} \sum_{i:y_i=p} (-f(x_i; \theta)^\top h_i) + \sum_{(i,j) \in \mathcal{N}^v} h_i^v h_j^v + \sum_{(i,j) \in \mathcal{N}} h_i^v P^v h_j^v \\
\leq \sum_{p=1}^{n_c} \sum_{i:y_i=p} (-c_{i}^p)^\top h_i + \sum_{(i,j) \in \mathcal{N}^v} h_i^v Q h_j^v + \sum_{(i,j) \in \mathcal{N}} h_i^v P^v h_j^v \\
+ \max \sum_{p=1}^{n_c} \sum_{i:y_i=p} (-c_{i}^p)^\top h_i \\
\leq M(\theta)
\]

Minimizing the upperbound in Equation (5) over \( h_1, \ldots, h_n \) is identical to minimizing the objective \( g(z_1, \ldots, z_{m}) \) in
Equation (4) since each example $j$ in class $i$ shares the same class mean embedding vector $c_i$. Absorbing the factor $m$ into the cost matrices i.e. $Q = mQ'$ and $P = mP'$, we arrive at the upperbound minimization problem defined in Equation (4). In the upperbound problem Equation (4), we consider the case where the pairwise cost matrices are diagonal matrices of non-negative values. Theorem 1 in the following subsection proves that finding the optimal solution of the upperbound problem in Equation (4) is equivalent to finding the minimum cost flow solution of the flow network $G'$ illustrated in Figure 2. Section B in the supplementary material shows the running time to compute the minimum cost flow (MCF) solution is approximately linear in $n_c$ and $d$. On average, it takes 24 ms and 53 ms to compute the MCF solution (discrete update) and to take a gradient descent step with npairs embedding [23] (network update), respectively on a machine with 1 TITAN-XP GPU and Xeon E5-2650.

4.2. Equivalence of the optimization problem to minimum cost flow

**Theorem 1.** The optimization problem in Equation (4) can be solved exactly in polynomial time by finding the minimum cost flow solution on the flow network $G'$.

**Proof.** Suppose we construct a vertex set $A = \{a_1, \ldots, a_n\}$ and partition $A$ into $\{A_{r}\}_{r=1}^t$ with the partition of $\{1, \ldots, n\}$ from equivalence relation $S^c$. Here, we will define $A_0$ as a union of subsets of size 1 (i.e. each element in $A_0$ is a singleton without a sibling), and $A_1, \ldots, A_t$ as the rest of the subsets (of size greater than or equal to 2).

Concretely, $|A| = n_c$ and $A = \bigcup_{r=0}^t A_r$.

Then, we construct $l + 1$ set of complete bipartite graphs $G_r = (A_r \cup B_r, E_r)$ where we define $g_r = |A_r|$ and $|B_r| = d \forall r$. Now suppose we construct a directed graph $G'$ by directing all edges $E_r$ from $A_r$ to $B_r$, attaching source $s$ to all vertices in $A_r$, and attaching sink $t$ to all vertices in $B_0$. Formally, $G' = \bigcup_{r=0}^t (A_r \cup B_r) \cup \{s, t\}, E'\}$.

The edges in $E'$ inherit all directed edges from source to vertices in $A_r$, edges from vertices in $B_0$ to sink, and $\{E_r\}_{r=0}^t$. We also attach $g_r$ number of edges for each vertex $b_r \in B_r$ to $b_0, q \in B_0$ and attach $n_c$ number of edges from each vertex $b_0, q \in B_0$ to $t$. Concretely, $E'$ is

\[
\{(s, a_p) | a_p \in A \} \cup \bigcup_{r=0}^t \cup_{i=0} g_r \cup \bigcup_{r=1} \{(b_r, b_0, q) \} \}_{r=0}^{r-1} \cup \{(b_0, t)\}_{c-1}^{c-1}.
\]

Edges incident to $s$ have capacity $u(s, a_p) = k_s$ and cost $v(s, a_p) = 0$ for all $a_p \in A$. The edges between $a_p \in A_r$ and $b_r \in B_r$ have capacity $u(a_p, b_r, q) = 1$ and cost $v(a_p, b_r, q) = -c_p[q]$. Each edge $i \in \{0, \ldots, g_r - 1\}$ between $b_r \in B_r$ and $b_0 \in B_0$ has capacity $u((b_r, b_0, q)) = 1$ and cost $u((b_r, b_0, q)) = 2\alpha_i$. Each edge $j \in \{0, \ldots, n_c - 1\}$ between $b_0, q \in B_0$ and $t$ has capacity $u((b_0, q, t)) = 1$ and cost $v((b_0, q, t)) = 2\beta_j$.

Figure 2 illustrates the flow network $G'$. The amount of flow from source to sink is $n_c k_s$. The figure omits the vertices in $A_0$ and the corresponding edges to $B_0$ to avoid the clutter.

Now we define the flow $\{f_v(e)\}_{e \in E'}$ for each edge indexed both by flow configuration $z_p \in \mathbb{R}^n$ where $z_p \in \{0, 1\}^d$. $||z_p||_1 = k_s \forall p$ and $e \in E'$ below in Equation (6).

\[\begin{align*}
(i) & \quad f_{s}(s, a_p) = k_s, \quad (ii) & \quad f_{s}(a_p, b_r, q) = z_p[q] \\
(iii) & \quad f_{e}((b_r, b_0, q)) = \begin{cases} 1 & \forall i < \sum_{p \in A_r} z_p[q] \\
0 & \text{otherwise} \end{cases} \\
(iv) & \quad f_{e}((b_0, q, t)) = \begin{cases} 1 & \forall j < \sum_{p=1}^{n_c} z_p[q] \\
0 & \text{otherwise} \end{cases}
\end{align*}\]

(6)

To prove the equivalence of computing the minimum cost flow solution and finding the minimum binary assignment in Equation (4), we need to show (1) that the flow defined in Equation (6) is feasible in $G'$ and (2) that the minimum cost flow solution of the network $G'$ and translating the computed flows to $\{z_p\}$ in Equation (4) indeed minimizes the discrete optimization problem. We first proceed with the flow feasibility proof.

It is easy to see the capacity constraints are satisfied by construction in Equation (6) so we prove that the flow conservation conditions are met at each vertices. First, the output flow from the source $\sum_{a_p \in A} f_{e}(s, a_p) = \sum_{p=1}^{n_c} k_s = n_c k_s$ is equal to the input flow. For each vertex $a_p \in A$, the amount of input flow is $k_s$ and the output flow is the same $\sum_{b_r \in B_r} f_{e}((a_p, b_r, q)) = \sum_{q=1}^{d} z_p[q] = ||z||_1 = k_s$.

For $r > 0$, for each vertex $b_r \in B_r$, denote the input flow as $y_{r, q} = \sum_{a_p \in A_r} f_{e}((a_p, b_r, q))$. The output flow is $\sum_{p=1}^{n_c} z_p[q] = y_{r, q}$ which is identical to the input flow. Therefore, the flow construction in Equation (6) is feasible in $G'$.

The last flow conservation condition is to check the connections from each vertex $b_0, q \in B_0$ to the sink. Denote the input flow at the vertex as $y_{0, q} = \sum_{p \in A_0} z_p[q] + \sum_{r=1}^{l} y_{r, q} = \sum_{p=1}^{n_c} z_p[q]$. The output flow is $\sum_{p=1}^{n_c} z_p[q] = y_{0, q}$ which is identical to the input flow. Therefore, the flow construction in Equation (6) is feasible in $G'$.

The second part of the proof is to check the optimality conditions and show the minimum cost flow finds the minimizer of Equation (4). Denote $\{f_v(e)\}_{e \in E'}$ as the minimum cost flow solution of the network $G'$ which minimizes the total cost $\sum_{e \in E'} v(f_v(e))$. Also denote the optimal flow from $a_p \in A$, to $b_r, q \in B_r$ as $z_p'[q]$. By optimality of the flow, $\sum_{e \in E'} v(f_v(e)) \leq \sum_{e \in E'} v(f_v(e)) \forall z$. By Lemma 1, the lhs of the inequality is equal to $\sum_{p=1}^{n_c} -c_p z_p' + \sum_{r=1}^{l} z_{p=1}^{n_c} z_{p=1}^{n_c} a z_p' + z_p' + z_{p=1}^{n_c} a z_p' + z_{p=1}^{n_c} a z_p' + z_{p=1}^{n_c} a z_p' + z_{p=1}^{n_c} a z_p'$.

\[11382\]
\[ \sum_{p_1 \neq p_2} c_p T z_{p}^T z_{p_2}. \]

Additionally, Lemma 2 shows the rhs of the inequality is equal to \( \sum_{p=1}^{n_c} c_p T z_p + \sum_{r=1}^{l} \sum_{p_1 \neq p_2 \in \{\rho|a_i \in A_r}\} \alpha z_{p_1} T z_{p_2} + \sum_{p_1 \neq p_2} \beta z_{p_1} T z_{p_2}. \)

Finally, \( \forall \{z\} \)

\[ \sum_{p=1}^{n_c} -c_p T z_p + \sum_{r=1}^{l} \sum_{p_1 \neq p_2 \in \{\rho|a_i \in A_r}\} \alpha z_{p_1} T z_{p_2} + \sum_{p_1 \neq p_2} \beta z_{p_1} T z_{p_2}. \]

This shows computing the minimum cost flow solution on \( G' \) and converting the flows to \( z \)'s, we can find the minimizer of the objective in Equation (4).

**Lemma 1.** Given the minimum cost flow \( \{f_e(e)\}_{e \in E'} \) of the network \( G' \), the total cost of the flow is \( \sum_{e \in E'} v(e) f_e(e) = \sum_{p=1}^{n_c} c_p T z_p + \sum_{r=1}^{l} \sum_{p_1 \neq p_2 \in \{\rho|a_i \in A_r\}} \alpha z_{p_1} T z_{p_2} + \sum_{p_1 \neq p_2} \beta z_{p_1} T z_{p_2}. \)

**Proof.** Proof in section A.2 of the supplementary material.

**Lemma 2.** Given a feasible flow \( \{f_e(e)\}_{e \in E'} \) of the network \( G' \), the total cost of the flow is \( \sum_{e \in E'} v(e) f_e(e) = \sum_{p=1}^{n_c} c_p T z_p + \sum_{r=1}^{l} \sum_{p_1 \neq p_2 \in \{\rho|a_i \in A_r\}} \alpha z_{p_1} T z_{p_2} + \sum_{p_1 \neq p_2} \beta z_{p_1} T z_{p_2}. \)

**Proof.** Proof in section A.2 of the supplementary material.

### 4.3. Learning the embedding representation given the hierarchical hash codes

Given a set of binary hash codes for the mean embeddings \( \{z^*_1, \ldots, z^*_n\}, \forall v = 1, \ldots, k \) computed from Equation (4), we can derive the hash codes for all \( n \) examples in the minibatch, \( h^v \ := \ z^*_p, \forall i : y_i = p \) and update the network weights \( \theta \) given the hierarchical hash codes in turn. The task is to update the embedding representations, \( \{f(x_i; \theta^v)\}_{i=1}^{n}, \forall v = 1, \ldots, k \), so that similar pairs of data have similar embedding representations indexed at the activated hash code dimensions and vice versa. Note, in terms of the hash code optimization in Equation (4) and the bound in Equation (5), this embedding update has the effect of tightening the bound gap \( M(\theta) \).

We employ the state of the art deep metric learning algorithms (denote as \( \ell_{\text{metric}}(\cdot) \)) such as triplet loss with semi-hard negative mining [21] and npairs loss [23] for this subproblem where the distance between two examples \( x_i \) and \( x_j \) at hierarchy level \( v \) is defined as \( d_{ij}^v = \| (h^v_i \lor h^v_j) \circ (f(x_i; \theta))^v - (f(x_j; \theta))^v \|_1 \). Utilizing the logical OR of the two binary masks, in contrast to independently indexing the representation with respective masks, to index the embedding representations helps prevent the pairwise distances frequently becoming zero due to the sparsity of the code. Note, this formulation in turn accommodates the backpropagation gradients to flow more easily. In our embedding representation learning subproblem, we need to learn the representations which respect the tree structural constraint on the corresponding hash code \( h = [h^1, \ldots, h^k] \in \{0, 1\}^{d \times k} \) where \( \|h^v\|_1 = 1, \forall v \neq k \) and \( \|h^k\|_1 = k_d \). To this end, we decompose the problem and compute the embedding loss per each hierarchy level \( v \) separately.

Furthermore, naively using the similarity labels to define similar pairs versus dissimilar pairs during the embedding learning subproblem could create a discrepancy between the hash code discrete optimization subproblem and the embedding learning subproblem leading to contradicting updates. Suppose two examples \( x_i \) and \( x_j \) are dissimilar and both had the highest activation at the same dimension \( o \) and the hash code for some level \( v \) was identical i.e. \( h^v_i[o] = h^v_j[o] = 1 \). Enforcing the metric learning loss with the class labels, in this case, would lead to increasing the highest activation for one example and decreasing the highest activation for the other example. This can be problematic for the example with decreased activation because it might get hashed to another occupied bucket after the gradient update and this can repeat.
Algorithm 1 Learning algorithm

\begin{verbatim}
input \( \theta^b \) (pretrained metric learning base model); \( \theta_d, k \)
initialize \( \theta_f = [\theta_b, \theta_d] \)
for \( t = 1, \ldots, \text{MAXITER} \) do
  Sample a minibatch \( \{x_i\} \) and initialize \( S_1^t = \emptyset \)
  for \( v = 1, \ldots, k \) do
    Update the flow network \( G' \) by computing class cost vectors 
    \( c^v_p = \frac{1}{m} \sum_i y_{i,v} f(x_i; \theta_f)^v \)
    Compute the hash codes \( \{h^v_i\} \) via minimum cost flow on \( G' \)
    Update \( S_2^{v+1} \) given \( S_2^v \) and \( \{h^v_i\} \)
    Remap the label to compute \( y^v \)
  end for
  Update the network parameter given the hash codes 
  \( \theta_f \leftarrow \theta_f - \eta(t) \partial_{\theta_f} \sum_{i=1}^k \ell_{\text{metric}}(\theta_f; h_{i,n_c}, y_{i,n_c}) \)
  Update stepsize \( \eta(t) \leftarrow \text{ADAM rule} [12] \)
end for
output \( \theta_f \) (final estimate);
\end{verbatim}

Network architecture

For fair comparison, we follow the protocol in [11] and use the NIN [15] architecture (denote the parameters \( \theta_h \)) with \textit{leaky relu} [30] with \( \tau = 5.5 \) as activation function and train Triplet embedding network with semi-hard negative mining [21]. Npairs network [23] from scratch as the base model, and snapshot the network weights (\( \theta^b \)) of the learned base model. Then we replace the last layer in (\( \theta^b \)) with a randomly initialized \( dk \) 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 learning procedure.

Hash table construction and query

We use the learned hash network \( \theta_f \) and apply Equation (1) to convert \( x_i \) into the hash code \( h(x_i; \theta_f) \) and use the base embedding network \( \theta^b \) to convert the data into the embedding representation \( f(x_i; \theta^b) \). Then, the embedding representation is hashed to buckets corresponding to the \( k_s \) set bits in the hash code. During inference, we convert a query data \( x_q \) into the hash code \( h(x_q; \theta_f) \) and into the embedding representation \( f(x_q; \theta^b) \). Once we retrieve the union of all bucket items indexed at the \( k_s \) set bits in the hash code, we apply a reranking procedure [27] based on the euclidean distance in the embedding space.

Evaluation metrics

Following the evaluation protocol in [11], we report our accuracy results using precision@k (Pr@k) and normalized mutual information (NMI) [17] 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 \( k_s = 1 \), treating each bucket as an individual cluster. 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.

6. Experiments

We report our results on Cifar-100 [13] and ImageNet [20] datasets and compare against several baseline methods. First baseline methods are the state of the art deep metric learning models [21, 23] performing an exhaustive linear search over the whole dataset given a query data (denote as ‘Metric’). Next baseline is the Binarization transform [1, 32] where the dimensions of the hash code corresponding to the top \( k_s \) dimensions of the embedding representation are set (denote as ‘Th’). Then we perform vector quantization [27] 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 \( k_s \) nearest centroids (denote as ‘VQ’). Another baseline is the quantizable representation in [11] (denote as [11]). In both Cifar-100 and ImageNet, we follow the data augmentation and preprocessing steps in [11] and train the metric learning base model with the same settings in [11] for fair comparison.

In Cifar-100 experiment, we set \( (d, k) = (32, 2) \) and \( (d, k) = (128, 2) \) for the npairs network and the triplet network, respectively. In ImageNet experiment, we set \( (d, k) = (512, 2) \) and \( (d, k) = (256, 2) \) for the npairs network and the triplet network, respectively. In ImageNetSplit experiment, we set \( (d, k) = (64, 2) \). We also perform LSH hashing [10] baseline and Deep Cauchy Hashing [6] baseline which both generate \( n \)-bit binary hash codes with \( 2^n \) buckets and compare against other methods when \( k_s = 1 \) (denote as ‘LSH’ and ‘DCH’, respectively). For the fair comparison, we set the number of buckets, \( 2^n = dk \).
We finetune the base model for 70 iterations when we optimize our methods. Table 1 shows the results from the triplet network and the Npairs network respectively. 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 the superior speedup over other baselines.

### Table 1: Results with Triplet network with hard negative mining and Npairs network. Querying test data against a hash table built on train set on Cifar-100.

| Method | SUF | Pr@1 | Pr@4 | Pr@16 | Test Set
|-------|-----|------|------|-------|---------
| k_m | Metric | SUF | Pr@1 | Pr@4 | Pr@16 |
| LSH | 1.00 | 9.00 | 9.39 | 7.45 | 113.21 | 11.70 | 8.90 | 5.50 |
| DCH | 10.07 | 9.82 | 8.43 | 6.44 | 202.50 | 13.87 | 11.77 | 8.99 |
| Th | 18.81 | 10.20 | 8.58 | 6.50 | 451.42 | 15.20 | 13.27 | 10.96 |
| VQ | 146.26 | 10.37 | 8.44 | 6.90 | 478.46 | 16.95 | 15.27 | 13.06 |
| Ours | 590.41 | 10.91 | 9.38 | 7.65 | 1174.35 | 17.00 | 15.53 | 13.54 |
| 2 | LSH | 4.90 | 56.84 | 56.01 | 5.84 | 11.36 | 14.61 | 14.90 |
| DCH | 86.11 | 68.88 | 65.55 | 50.00 | 649.74 | 65.68 | 65.00 |
| Th | 97.12 | 57.30 | 57.30 | 11.07 | 63.60 | 63.00 | 62.00 |
| VQ | 146.26 | 10.37 | 8.44 | 6.90 | 478.46 | 16.95 | 15.27 | 13.06 |
| Ours | 590.41 | 10.91 | 9.38 | 7.65 | 1174.35 | 17.00 | 15.53 | 13.54 |
| 3 | Th | 97.12 | 57.30 | 57.30 | 11.07 | 63.60 | 63.00 | 62.00 |
| VQ | 146.26 | 10.37 | 8.44 | 6.90 | 478.46 | 16.95 | 15.27 | 13.06 |
| Ours | 590.41 | 10.91 | 9.38 | 7.65 | 1174.35 | 17.00 | 15.53 | 13.54 |

### Table 2: Results with Triplet network with hard negative mining and Npairs [23] Network. Querying ImageNet val data against hash table built on val set.

### 6.1. Cifar-100

Cifar-100 [13] dataset has 100 classes. Each class has 500 images for train and 100 images for test. Given a query image from train, we experiment the search performance both when the hash table is constructed from train and from test. The batch size is set to 128 in Cifar-100 experiment. We finetune the base model for 70k iterations and decayed the learning rate to 0.3 of previous learning rate after 20k iterations when we optimize our methods. Table 1 shows the results from the triplet network and the Npairs network respectively.

### Table 3: Hash table NMI for Cifar-100 and Imagenet.

| Method | SUF | Pr@1 | Pr@4 | Pr@16 | Train Set
|-------|-----|------|------|-------|---------
| k_m | Metric | SUF | Pr@1 | Pr@4 | Pr@16 | 1174.35 | 17.00 | 15.53 | 13.54 |
| LSH | 1.00 | 9.00 | 9.39 | 7.45 | 113.21 | 11.70 | 8.90 | 5.50 |
| DCH | 10.07 | 9.82 | 8.43 | 6.44 | 202.50 | 13.87 | 11.77 | 8.99 |
| Th | 18.81 | 10.20 | 8.58 | 6.50 | 451.42 | 15.20 | 13.27 | 10.96 |
| VQ | 146.26 | 10.37 | 8.44 | 6.90 | 478.46 | 16.95 | 15.27 | 13.06 |
| Ours | 590.41 | 10.91 | 9.38 | 7.65 | 1174.35 | 17.00 | 15.53 | 13.54 |
| 2 | LSH | 4.90 | 56.84 | 56.01 | 5.84 | 11.36 | 14.61 | 14.90 |
| DCH | 86.11 | 68.88 | 65.55 | 50.00 | 649.74 | 65.68 | 65.00 |
| Th | 97.12 | 57.30 | 57.30 | 11.07 | 63.60 | 63.00 | 62.00 |
| VQ | 146.26 | 10.37 | 8.44 | 6.90 | 478.46 | 16.95 | 15.27 | 13.06 |
| Ours | 590.41 | 10.91 | 9.38 | 7.65 | 1174.35 | 17.00 | 15.53 | 13.54 |
| 3 | Th | 97.12 | 57.30 | 57.30 | 11.07 | 63.60 | 63.00 | 62.00 |
| VQ | 146.26 | 10.37 | 8.44 | 6.90 | 478.46 | 16.95 | 15.27 | 13.06 |
| Ours | 590.41 | 10.91 | 9.38 | 7.65 | 1174.35 | 17.00 | 15.53 | 13.54 |

### Table 4: Results with Triplet network with hard negative mining. Querying ImageNet val set in C-test against hash table built on val set in C-train.

### 6.2. ImageNet

ImageNet ILSVRC-2012 [20] 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.
Figure 3: Visualization of the examples mapped by our trained three level hash codes \([h^{(1)}, h^{(2)}]\) on Cifar-100. Each parent node (denoted as depth 1) is color coded in red, yellow, blue, and green in \(cw\) order. Each color coded box (denoted as depth 2) shows examples of the hashed items in each child node.


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 [8]. We finetune npairs base model and triplet base model as in [11] and add a randomly initialized fully connected layer to learn hierarchical representation. Then, we train the parameters in the newly added layer with other parameters fixed. When we train with npairs loss, we set the batch size to 1024 and train for 15k iterations decaying the learning rate to 0.3 of previous learning rate after each 6k iterations. Also, when we train with triplet loss, we set the batch size to 512 and train for 30k iterations decaying the learning rate of 0.3 of previous learning rate after each 10k iterations. Our results in Table 2 show that our method outperforms the state of the art deep metric learning base models in search accuracy while providing up to 1298× speedup over exhaustive linear search. Table 3 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 the base metric learning methods.

6.3. ImageNetSplit

In order to test the generalization performance of our learned representation against previously unseen classes, we performed an experiment on ImageNet where the set of classes for training and testing are completely disjoint. Each class in ImageNet ILSVRC-2012 [20] dataset has super-class based on WordNet [18]. We select 119 super-classes which have exactly two sub-classes in 1000 classes of ImageNet ILSVRC-2012 dataset. Then, we split the two sub-classes of each 119 super-class into \(C_{train}\) and \(C_{test}\), where \(C_{train} \cap C_{test} = \emptyset\). Section D in the supplementary material shows the class names in \(C_{train}\) and \(C_{test}\). We use the images downsampled to \(32 \times 32\) from [8]. We train the models with triplet embedding on \(C_{train}\) and test the models on \(C_{test}\). The batch size is set to 200 in ImageNetSplit dataset. We finetune the base model for 50k iterations and decayed the learning rate to 0.3 of previous learning rate after 40k iterations when we optimize our methods. We also perform vector quantization with the centroids obtained from \(C_{train}\) (denote as ‘VQ-train’) and \(C_{test}\) (denote as ‘VQ-test’), respectively. Table 4 shows our method preserves the accuracy without compromising the speedup factor.

Note, in all our experiments in Tables 1 to 4, while all the baseline methods show severe degradation in the speedup over the code compound parameter \(k_s\), the results show that the proposed method robustly withstands the speedup degradation over \(k_s\). This is because our method 1) greatly increases the quantization granularity beyond other baseline methods and 2) hashes the items more uniformly over the buckets. In effect, indexing multiple buckets in our quantized representation does not as adversarially effect the search speedup as other baselines. Figure 3 shows a qualitative result with npairs network on Cifar-100, where \(d = 32, k = 2, k_s = 1\). As an interesting side effect, our qualitative result indicates that even though our method does not use any super/sub-class labels or the entire label information during training, optimizing for the objective in Equation (2) naturally discovers and organizes the data exhibiting a meaningful hierarchy where similar subclasses share common parent nodes.

7. Conclusion

We have shown a novel end-to-end learning algorithm where the quantization granularity is significantly increased via hierarchically quantized representations while preserving the search accuracy and maintaining the computational complexity practical for the mini-batch stochastic gradient descent setting. This not only provides the state of the art accuracy results but also unlocks significant improvement in inference speedup providing the highest reported inference speedup on Cifar100 and ImageNet datasets respectively.

Acknowledgements

This work was partially supported by Kakao, Kakao Brain and Basic Science Research Program through the National Research Foundation of Korea (NRF) (2017R1E1A1A01077431). Hyun Oh Song is the corresponding author.
References


[23] K. Sohn. Improved deep metric learning with multi-class n-pair loss objective. In NIPS, 2016. 1, 3, 4, 5, 6, 7


