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

[22], zero-shot learning [31, 5], video representation learning [28], and similarity-based interior design [2], etc.

Another line of research focuses on learning binary hamming ranking [29, 33, 19, 14] representations via neural networks. Although comparing binary hamming codes is more efficient than comparing continuous embedding representations, 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 representation and the corresponding sparsity constrained binary hash code in an efficient mini-batch based end-to-end learning framework. Jeong & Song [11] motivated 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 constraint, but at the same time, greatly increasing the number of representable buckets by imposing an efficient hierarchical structure on the hash code to unlock significant improvement in the speedup factor.

3. Problem formulation

Consider the following hash function

$$r(\mathbf{x}) = \underset{\mathbf{h}\in\{0,1\}^d}{\operatorname{argmin}} -f(\mathbf{x};\boldsymbol{\theta})^\intercal \mathbf{h}$$

under the constraint that $\|\mathbf{h}\|_1 = k_s$. The idea is to optimize the weights in the neural network $f(\cdot;\boldsymbol{\theta}) : \mathcal{X} \to \mathbb{R}^d$, take k_s highest activation dimensions, activate the corresponding dimensions in the binary compound hash code $\mathbf{h}$, and hash the data $\mathbf{x} \in \mathcal{X}$ into the corresponding active buckets of a hash table $\mathcal{H}$. During inference, a query $\mathbf{x}_q$ is given, and all the hashed items in the k_s active bits set by the hash function $r(\mathbf{x}_q)$ are retrieved as the candidate nearest items. Often times [27], these candidates are reranked based on the euclidean distance in the base embedding representation $f(\cdot;\boldsymbol{\theta})$ space.

Given a query $\mathbf{h}_q$, the expected number of retrieved items is $\sum_{i\neq q}\Pr(\mathbf{h}_i^\intercal \mathbf{h}_q \neq 0)$. 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 $(\Pr(\mathbf{h}_i^\intercal \mathbf{h}_q \neq 0))^{-1} = (1 - \binom{d-k_s}{k_s}/\binom{d}{k_s})^{-1}$. In case $d \gg k_s$, this ratio approaches $d/{k_s}^2$.

Now, suppose we design a hash function $r(\mathbf{x})$ so that the function has total $\dim(r(\mathbf{x})) = d^k$ (*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 number 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 efficient tree structure. First, we use $\dim(f(\mathbf{x})) = dk$ 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 $\widetilde{\mathbf{h}} = [\mathbf{h}^1, \dots, \mathbf{h}^k] \in \{0,1\}^{d\times k}$ where $\|\mathbf{h}^v\|_1 = 1 \ \forall v \neq k$ and $\|\mathbf{h}^k\|_1 = k_s$. The superscript denotes the level index in the hierarchy. Now, suppose we construct a tree $\mathcal{T}$ with branching factor d, depth k where the root node has the level index of 0. Let each d^k leaf node in $\mathcal{T}$ represent a bucket indexed by the hash function $r(\mathbf{x})$. Then, we can interpret each $\mathbf{h}^v$ vector to indicate the branching from depth $v-1$ to depth v in $\mathcal{T}$. Note, from the construction of $\widetilde{\mathbf{h}}$, 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 $r(\mathbf{x}) : \mathbb{R}^{d\times k} \to \{0,1\}^{d^k}$ can be expressed compactly as Equation (1).

$$r(\mathbf{x}) = \bigotimes_{v=1}^{k} \underset{\mathbf{h}^v}{\operatorname{argmin}} - (f(\mathbf{x};\boldsymbol{\theta})^v)^\intercal \, \mathbf{h}^v \tag{1}$$

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

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

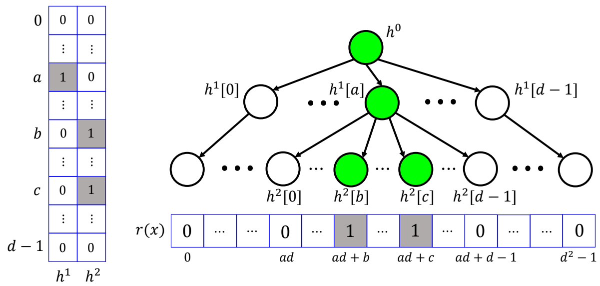

Figure 1: Example hierarchical structure for $k = 2$ and $k_s = 2$. (Left) The hash code for each embedding representation $[f(\mathbf{x}_i;\boldsymbol{\theta})^1, f(\mathbf{x}_i;\boldsymbol{\theta})^2] \in \mathbb{R}^{2d}$. (Right) Corresponding activated hash buckets out of total d^2 buckets.

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 $\{f(\mathbf{x}_i;\boldsymbol{\theta})\}_{i=1}^n$, we wish to compute the optimal binary hash code $\{\mathbf{h}_1,\ldots,\mathbf{h}_n\}$ 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 $\mathbf{x}_i$ and $\mathbf{x}_j$ 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, $\mathbf{h}_i^v$ and $\mathbf{h}_j^v$ 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 $\{\mathbf{h}_1^1,\ldots,\mathbf{h}_n^1\}$ to the leaf nodes $\{\mathbf{h}_1^k,\ldots,\mathbf{h}_n^k\}$ as shown in Equation (2).

$$\underset{\mathbf{h}^k_{1:n},\ldots,\mathbf{h}^1_{1:n}}{\text{minimize}} \quad \underbrace{\sum_{v=1}^{k}\sum_{i=1}^{n} -(f(\mathbf{x}_i;\boldsymbol{\theta})^v)^\intercal\, \mathbf{h}_i^v}_{\text{unary term}} \tag{2}$$
$$+ \underbrace{\sum_{v=2}^{k}\sum_{(i,j)\in\mathcal{N}} \mathbf{h}_i^{v\intercal} Q' \mathbf{h}_j^v \prod_{w=1}^{v-1} \mathbb{1}(\mathbf{h}_i^w = \mathbf{h}_j^w)}_{\text{sibling penalty}} + \underbrace{\sum_{v=1}^{k}\sum_{(i,j)\in\mathcal{N}} \mathbf{h}_i^{v\intercal} P' \mathbf{h}_j^v}_{\text{orthogonality}}$$
$$\text{subject to} \quad \|\mathbf{h}_i^v\| = \begin{cases} 1 & \forall v \neq k \\ k_s & v = k \end{cases},\ \mathbf{h}_i^v \in \{0,1\}^d,\ \forall i,$$

where $\mathcal{N}$ denotes the set of dissimilar pairs of data and $\mathbb{1}(\cdot)$ 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[1] $v \in \{1,\ldots,k\}$. The unary term in the objective encourages selecting as large elements of each 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.

$$\underset{\mathbf{h}_1,\ldots,\mathbf{h}_n}{\text{minimize}} \quad \underbrace{\sum_{i=1}^{n} -(f(\mathbf{x}_i;\boldsymbol{\theta})^v)^\intercal\, \mathbf{h}_i}_{\text{unary term}} + \underbrace{\sum_{(i,j)\in\mathcal{S}^v} \mathbf{h}_i^\intercal Q' \mathbf{h}_j}_{\text{sibling penalty}} + \underbrace{\sum_{(i,j)\in\mathcal{N}} \mathbf{h}_i^\intercal P' \mathbf{h}_j}_{\text{orthogonality}} \tag{3}$$

where $\mathcal{S}^v = \{(i,j)\in\mathcal{N} \mid \mathbf{h}_i^w = \mathbf{h}_j^w,\ \forall w = 1,\ldots,v-1\}$ denotes the set of pairs of siblings at level v in $\mathcal{T}$, and Q', P' 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 $k_s = 1, k = 1, d > 2$ [3, 11]. Inspired by [11], we use the average embedding of each class within the minibatch $\mathbf{c}_p^v = \frac{1}{m}\sum_{i:y_i=p} f(\mathbf{x}_i;\boldsymbol{\theta})^v \in \mathbb{R}^d$ as shown in Equation (4).

$$\underset{\mathbf{z}_1,\ldots,\mathbf{z}_{n_c}}{\text{minimize}} \quad \underbrace{\sum_{p=1}^{n_c} -(\mathbf{c}_p^v)^\intercal \mathbf{z}_p + \sum_{\substack{(p,q)\in\mathcal{S}_z^v \\ p\neq q}} {\mathbf{z}_p}^\intercal Q \mathbf{z}_q + \sum_{p\neq q} {\mathbf{z}_p}^\intercal P \mathbf{z}_q}_{:=\hat{g}(\mathbf{z}_1,\ldots,\mathbf{z}_{n_c})}$$
$$\text{subject to} \quad \|\mathbf{z}_p\| = \begin{cases} 1 & \forall v \neq k \\ k_s & v = k \end{cases},\ \mathbf{z}_p \in \{0,1\}^d,\ \forall p, \tag{4}$$

where $\mathcal{S}_z^v = \{(p,q) \mid \mathbf{z}_p^w = \mathbf{z}_q^w,\ \forall w = 1,\ldots,v-1\}$, 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 $g(\cdot;\boldsymbol{\theta})$) by a gap $M(\boldsymbol{\theta})$ which depends only on $\boldsymbol{\theta}$. Concretely, rewriting the summation in the unary term in g, we get

$$\begin{aligned} g(\mathbf{h}_1,\ldots,\mathbf{h}_n;\boldsymbol{\theta}) &= \sum_{p}^{n_c}\sum_{i:y_i=p} -(f(\mathbf{x}_i;\boldsymbol{\theta})^v)^\intercal\, \mathbf{h}_i \\ &\quad + \sum_{(i,j)\in\mathcal{S}^v} \mathbf{h}_i^\intercal Q' \mathbf{h}_j + \sum_{(i,j)\in\mathcal{N}} \mathbf{h}_i^\intercal P' \mathbf{h}_j \\ &\leq \sum_{p}^{n_c}\sum_{i:y_i=p} -(\mathbf{c}_p^v)^\intercal\, \mathbf{h}_i + \sum_{(i,j)\in\mathcal{S}^v} \mathbf{h}_i^\intercal Q' \mathbf{h}_j + \sum_{(i,j)\in\mathcal{N}} \mathbf{h}_i^\intercal P' \mathbf{h}_j \\ &\quad + \underbrace{\underset{\hat{\mathbf{h}}_1,\ldots,\hat{\mathbf{h}}_n}{\text{maximize}} \sum_{p}^{n_c}\sum_{i:y_i=p} (\mathbf{c}_p^v - f(\mathbf{x}_i;\boldsymbol{\theta})^v)^\intercal\, \hat{\mathbf{h}}_i}_{:=M(\boldsymbol{\theta})}. \end{aligned} \tag{5}$$

Minimizing the upperbound in Equation (5) over $\mathbf{h}_1,\ldots,\mathbf{h}_n$ is identical to minimizing the objective $\hat{g}(\mathbf{z}_1,\ldots,\mathbf{z}_{n_c})$ in

[1] In Equation (3), we omit the dependence of v for all $\mathbf{h}_1,\ldots,\mathbf{h}_n$ to avoid the notation clutter.

Equation (4) since each example j in class i shares the same class mean embedding vector $\mathbf{c}_i$. 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 $A = \{a_1, \ldots, a_{n_c}\}$ and partition A into $\{A_r\}_{r=0}^{l}$ with the partition of $\{1, \ldots, n_c\}$ from equivalence relation $\mathcal{S}_z^v$[2]. 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 $A_1, \ldots, A_l$ as the rest of the subsets (of size greater than or equal to2). Concretely, $|A| = n_c$ and $A = \bigcup_{r=0}^{l} A_r$.

Then, we construct $l+1$ set of complete bipartite graphs $\{G_r = (A_r \cup B_r, E_r)\}_{r=0}^{l}$ 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, $G' = \left(\bigcup_{r=0}^{l} (A_r \cup B_r) \cup \{s, t\}, E'\right)$. The edges in E' inherit all directed edges from source to vertices in A_r, edges from vertices in B_0 to sink, and $\{E_r\}_{r=0}^{l}$. We also attach g_r number of edges for each vertex $b_{r,q} \in B_r$ 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

$$\{(s, a_p) | a_p \in A\} \cup \bigcup_{r=0}^{l} E_r \cup \bigcup_{r=1}^{l} \{(b_{r,q}, b_{0,q})_i\}_{i=0}^{g_r-1} \cup \{(b_{0,q}, t)_j\}_{j=0}^{n_c-1}.$$

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 $b_{r,q} \in B_r$ have capacity $u(a_p, b_{r,q}) = 1$ and cost $v(a_p, b_{r,q}) = -\mathbf{c}_p[q]$. Each edge $i \in \{0, \ldots, g_r - 1\}$ between $b_{r,q} \in B_r$ and $b_{0,q} \in B_0$ has capacity $u\left((b_{r,q}, b_{0,q})_i\right) = 1$ and cost $u\left((b_{r,q}, b_{0,q})_i\right) = 2\alpha i$. Each edge $j \in \{0, \ldots, n_c - 1\}$ between $b_{0,q} \in B_0$ and t has capacity $u\left((b_{0,q}, t)_j\right) = 1$ and cost $v\left((b_{0,q}, t)_j\right) = 2\beta j$.

[2]Define $(p, q) \in \mathcal{S}_z^v \iff a_p, a_q \in A_r, \forall r \geq 1$

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 $\{f_z(e)\}_{e \in E'}$ for each edge indexed both by flow configuration $\mathbf{z}_p \in \mathbf{z}_{1:n_c}$ where $\mathbf{z}_p \in \{0,1\}^d$, $\|\mathbf{z}_p\|_1 = k_s\ \forall p$ and $e \in E'$ below in Equation (6).

$$(i)\ f_z(s, a_p) = k_s,\ (ii)\ f_z(a_p, b_{r,q}) = \mathbf{z}_p[q]$$
$$(iii)\ f_z\left((b_{r,q}, b_{0,q})_i\right) = \begin{cases} 1 & \forall i < \sum_{p: a_p \in A_r} \mathbf{z}_p[q] \\ 0 & \text{otherwise} \end{cases}$$
$$(iv)\ f_z\left((b_{0,q}, t)_j\right) = \begin{cases} 1 & \forall j < \sum_{p=1}^{n_c} \mathbf{z}_p[q] \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

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 $\{\mathbf{z}_p\}$ 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 $\sum_{a_p \in A} f_z(s, a_p) = \sum_{p=1}^{n_c} k_s = n_c k_s$ 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 $\sum_{b_{r,q} \in B_r} f_z(a_p, b_{r,q}) = \sum_{q=1}^{d} \mathbf{z}_p[q] = \|\mathbf{z}\|_1 = k_s$.

For $r > 0$, for each vertex $b_{r,q} \in B_r$, denote the input flow as $y_{r,q} = \sum_{a_p \in A_r} f_z(a_p, b_{r,q}) = \sum_{p: a_p \in A_r} \mathbf{z}_p[q]$. The output flow is $\sum_{i=0}^{g_r-1} f_z((b_{r,q}, b_{0,q})_i) = \sum_{p: a_p \in A_r} \mathbf{z}_p[q] = y_{r,q}$. The second term vanishes because of Equation (6) (iii).

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 $y_{0,q} = \sum_{p: a_p \in A_0} \mathbf{z}_p[q] + \sum_{r=1}^{l} y_{r,q} = \sum_{p=1}^{n_c} \mathbf{z}_p[q]$. The output flow is $\sum_{j=0}^{n_c-1} f_z((b_{0,q}, t)_j) = \sum_{p=1}^{n_c} \mathbf{z}_p[q] = y_{0,q}$ 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, $\{f_o(e)\}_{e \in E'}$ as the minimum cost flow solution of the network G' which minimizes the total cost $\sum_{e \in E'} v(e) f_o(e)$. Also denote the optimal flow from $a_p \in A_r$ to $b_{r,q} \in B_r$, $f_o(a_p, b_q)$ as $\mathbf{z}'_p[q]$. By optimality of the flow, $\{f_o(e)\}_{e \in E'}$, $\sum_{e \in E'} v(e) f_o(e) \leq \sum_{e \in E'} v(e) f_z(e)\quad \forall z$. By Lemma 1, the *lhs* of the inequality is equal to $\sum_{p=1}^{n_c} -\mathbf{c}_p{}^T \mathbf{z}'_p + \sum_{r=1}^{l} \sum_{p_1 \neq p_2 \in \{p | a_p \in A_r\}} \alpha \mathbf{z}'_{p_1}{}^T \mathbf{z}'_{p_2} +$

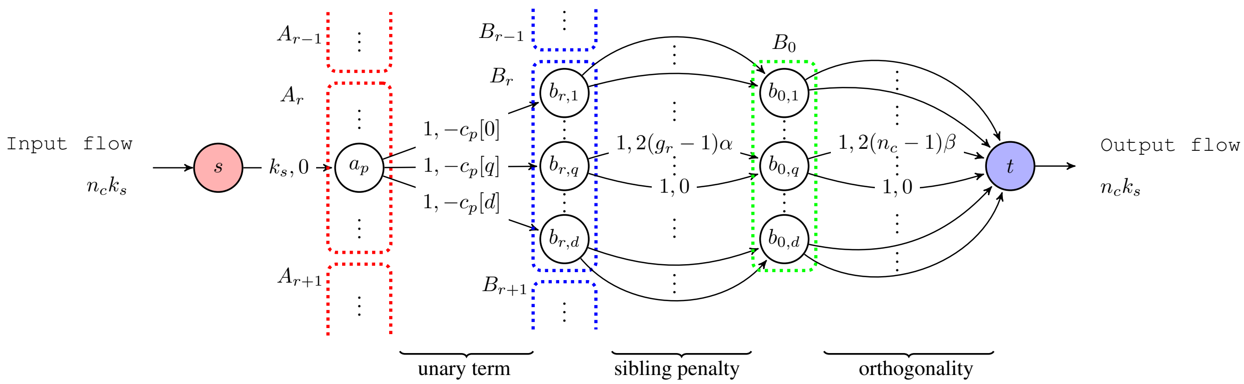

Figure 2: Equivalent flow network diagram G' corresponding to the discrete optimization Equation (4). Edge labels show the capacity and the cost respectively.

$\sum_{p_1 \neq p_2} \beta {\mathbf{z}'_{p_1}}^T \mathbf{z}'_{p_2}$. Additionally, Lemma 2 shows the *rhs* of the inequality is equal to $\sum_{p=1}^{n_c} -{\mathbf{c}_p}^T \mathbf{z}_p + \sum_{r=1}^{l} \sum_{p_1 \neq p_2 \in \{p | a_p \in A_r\}} \alpha {\mathbf{z}_{p_1}}^T \mathbf{z}_{p_2} + \sum_{p_1 \neq p_2} \beta {\mathbf{z}_{p_1}}^T \mathbf{z}_{p_2}$. Finally, $\forall \{\mathbf{z}\}$

$$\sum_{p=1}^{n_c} -{\mathbf{c}_p}^T {\mathbf{z}'}_p + \sum_{r=1}^{l} \sum_{p_1 \neq p_2 \in \{p | a_p \in A_r\}} \alpha {\mathbf{z}'_{p_1}}^T \mathbf{z}'_{p_2} + \sum_{p_1 \neq p_2} \beta {\mathbf{z}'_{p_1}}^T \mathbf{z}'_{p_2}$$
$$\leq \sum_{p=1}^{n_c} -{\mathbf{c}_p}^T \mathbf{z}_p + \sum_{r=1}^{l} \sum_{p_1 \neq p_2 \in \{p | a_p \in A_r\}} \alpha {\mathbf{z}_{p_1}}^T \mathbf{z}_{p_2} + \sum_{p_1 \neq p_2} \beta {\mathbf{z}_{p_1}}^T \mathbf{z}_{p_2}.$$

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

Lemma 1. *Given the minimum cost flow $\{f_o(e)\}_{e \in E'}$ of the network G', the total cost of the flow is $\sum_{e \in E'} v(e) f_o(e) = \sum_{p=1}^{n_c} -{\mathbf{c}_p}^T {\mathbf{z}'}_p + \sum_{r=1}^{l} \sum_{p_1 \neq p_2 \in \{p | a_p \in A_r\}} \alpha {\mathbf{z}'_{p_1}}^T \mathbf{z}'_{p_2} + \sum_{p_1 \neq p_2} \beta {\mathbf{z}'_{p_1}}^T \mathbf{z}'_{p_2}$.*

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

Lemma 2. *Given a feasible flow $\{f_z(e)\}_{e \in E'}$ of the network G', the total cost of the flow is $\sum_{e \in E'} v(e) f_z(e) = \sum_{p=1}^{n_c} -{\mathbf{c}_p}^T \mathbf{z}_p + \sum_{r=1}^{l} \sum_{p_1 \neq p_2 \in \{p | a_p \in A_r\}} \alpha {\mathbf{z}_{p_1}}^T \mathbf{z}_{p_2} + \sum_{p_1 \neq p_2} \beta {\mathbf{z}_{p_1}}^T \mathbf{z}_{p_2}$.*

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 $\{\mathbf{z}_1^v, \ldots, \mathbf{z}_{n_c}^v\}$, $\forall v = 1, \ldots, k$ computed from Equation (4), we can derive the hash codes for all n examples in the minibatch, $\mathbf{h}_i^v := \mathbf{z}_p^v \;\; \forall i : y_i = p$ and update the network weights $\boldsymbol{\theta}$ given the hierarchical hash codes in turn. The task is to update the embedding representations, $\{f(\mathbf{x}_i; \boldsymbol{\theta})^v\}_{i=1}^n$, $\forall v = 1, \ldots, k$, 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 $M(\boldsymbol{\theta})$.

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 $\mathbf{x}_i$ and $\mathbf{x}_j$ at hierarchy level v is defined as $d_{ij}^v = \| \left(\mathbf{h}_i^v \vee \mathbf{h}_j^v\right) \odot (f(\mathbf{x}_i; \boldsymbol{\theta})^v - f(\mathbf{x}_j; \boldsymbol{\theta})^v) \|_1$. 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 $\mathbf{h} = [\mathbf{h}^1, \ldots, \mathbf{h}^k] \in \{0,1\}^{d \times k}$ where $\|\mathbf{h}^v\|_1 = 1 \;\; \forall v \neq k$ and $\|\mathbf{h}^k\|_1 = k_s$. 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 $\mathbf{x}_i$ and $\mathbf{x}_j$ 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.* $\mathbf{h}_i^v[o] = \mathbf{h}_j^v[o] = 1$. 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 causing instability in the optimization process.

However, if we relabel the two examples so that they are treated as the same class as long as they have the same hash code at the level, the update wouldn't decrease the activations for any example, and the sibling term (the second term) in Equation (4) would automatically take care of splitting the two examples in the next subsequent levels.

To this extent, we apply *label remapping* as follows. $y_i^v = remap(\mathbf{h}_i^v)$, where $remap(\cdot)$ assigns arbitrary unique labels to each unique configuration of $\mathbf{h}_i^v$. Concretely, $remap(\mathbf{h}_i^v) = remap(\mathbf{h}_j^v) \iff y_i^v = y_j^v$. Finally, the embedding representation learning subproblem aims to solve Equation (7) given the hash codes and the remapped labels. Section C in the supplementary material includes the ablation study of label remapping.

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \sum_{v=1}^{k} \ell_{\text{metric}} \left(\{f(x_i;\boldsymbol{\theta})^v\}_{i=1}^n;\ \{\mathbf{h}_i^v\}_{i=1}^n, \{y_i^v\}_{i=1}^n \right) \quad (7)$$

Following the protocol in [11], we use the Tensorflow implementation of deep metric learning algorithms in `tf.contrib.losses.metric_learning`.

5. Implementation details

Algorithm 1 Learning algorithm

input $\boldsymbol{\theta}_b^{\text{emb}}$ (pretrained metric learning base model); $\boldsymbol{\theta}_d$, k
initialize $\boldsymbol{\theta}_f = [\boldsymbol{\theta}_b, \boldsymbol{\theta}_d]$
  for $t = 1, \ldots,$ MAXITER **do**
    Sample a minibatch $\{\mathbf{x}_i\}$ and initialize $\mathcal{S}_z^1 = \emptyset$
    for $v = 1, \cdots, k$ **do**
      Update the flow network G' by computing class cost vectors $\mathbf{c}_p^v = \frac{1}{m} \sum_{i:y_i=p} f(\mathbf{x}_i;\boldsymbol{\theta}_f)^v$
      Compute the hash codes $\{\mathbf{h}_i^v\}$ via minimum cost flow on G'
      Update $\mathcal{S}_z^{v+1}$ given $\mathcal{S}_z^v$ and $\{h_i^v\}$
      Remap the label to compute y^v
    end for
    Update the network parameter given the hash codes

$$\boldsymbol{\theta}_f \leftarrow \boldsymbol{\theta}_f - \eta^{(t)} \partial_{\boldsymbol{\theta}_f} \sum_{v=1}^{k} \ell_{\text{metric}}(\boldsymbol{\theta}_f;\ \mathbf{h}_{1:n_c}^v, y_{1:n_c}^v)$$

    Update stepsize $\eta^{(t)} \leftarrow$ ADAM rule [12]
  end for
output $\boldsymbol{\theta}_f$ (final estimate);

Network architecture For fair comparison, we follow the protocol in [11] and use the NIN [15] architecture (denote the parameters $\boldsymbol{\theta}_b$) with *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 ($\boldsymbol{\theta}_b^{\text{emb}}$) of the learned base model. Then we replace the last layer in ($\boldsymbol{\theta}_b^{\text{emb}}$) with a randomly initialized dk dimensional fully connected projection layer ($\boldsymbol{\theta}_d$) and finetune the hash network (denote the parameters as $\boldsymbol{\theta}_f = [\boldsymbol{\theta}_b, \boldsymbol{\theta}_d]$). Algorithm 1 summarizes the learning procedure.

Hash table construction and query We use the learned hash network $\boldsymbol{\theta}_f$ and apply Equation (1) to convert $\mathbf{x}_i$ into the hash code $\mathbf{h}(\mathbf{x}_i;\boldsymbol{\theta}_f)$ and use the base embedding network $\boldsymbol{\theta}_b^{\text{emb}}$ to convert the data into the embedding representation $f(\mathbf{x}_i;\boldsymbol{\theta}_b^{\text{emb}})$. 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 $\mathbf{x}_q$ into the hash code $\mathbf{h}(\mathbf{x}_q;\boldsymbol{\theta}_f)$ and into the embedding representation $f(\mathbf{x}_q;\boldsymbol{\theta}_b^{\text{emb}})$. 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 Cfar-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, $2^n = dk$.

		Triplet								Npairs							
		test				*train*				*test*				*train*			
	Method	SUF	Pr@1	Pr@4	Pr@16	SUF	Pr@1	Pr@4	Pr@16	SUF	Pr@1	Pr@4	Pr@16	SUF	Pr@1	Pr@4	Pr@16
k_s	Metric	1.00	56.78	55.99	53.95	1.00	62.64	61.91	61.22	1.00	57.05	55.70	53.91	1.00	61.78	60.63	59.73
1	LSH	**138.83**	52.52	48.67	39.71	**135.64**	60.45	58.10	54.00	29.74	53.55	50.75	43.03	30.75	59.87	58.34	55.35
	DCH	96.13	56.26	55.65	54.26	89.60	61.06	60.80	60.81	41.59	57.23	56.25	54.45	40.49	61.59	60.77	60.12
	Th	41.21	54.82	52.88	48.03	43.19	61.56	60.24	58.23	12.72	54.95	52.60	47.16	13.65	60.80	59.49	57.27
	VQ	22.78	56.74	55.94	53.77	40.35	62.54	61.78	60.98	34.86	56.76	55.35	53.75	31.35	61.22	60.24	59.34
	[11]	97.67	57.63	57.16	55.76	97.77	63.85	63.40	63.39	54.85	58.19	57.22	55.87	54.90	**63.11**	62.29	61.94
	Ours	97.67	**58.42**	**57.88**	**56.58**	97.28	**64.73**	**64.63**	**64.69**	**101.1**	**58.28**	**57.79**	**56.92**	**97.47**	63.06	**62.62**	**62.44**
2	Th	14.82	56.55	55.62	52.90	15.34	62.41	61.68	60.89	5.09	56.52	55.28	53.04	5.36	61.65	60.50	59.50
	VQ	5.63	56.78	56.00	53.99	6.94	62.66	61.92	61.26	6.08	57.13	55.74	53.90	5.44	61.82	60.56	59.70
	[11]	76.12	57.30	56.70	55.19	78.28	63.60	63.19	63.09	16.20	57.27	55.98	54.42	16.51	61.98	60.93	60.15
	Ours	**98.38**	**58.39**	**57.51**	**56.09**	**97.20**	**64.35**	**63.91**	**63.81**	**69.48**	**57.60**	**56.98**	**55.82**	**69.91**	**62.19**	**61.71**	**61.27**
3	Th	7.84	56.78	55.91	53.64	8.04	62.66	61.88	61.16	3.10	56.97	55.56	53.76	3.21	61.75	60.66	59.73
	VQ	2.83	56.78	55.99	53.95	2.96	62.62	61.92	61.22	2.66	57.01	55.69	53.90	2.36	61.78	60.62	59.73
	[11]	42.12	56.97	56.25	54.40	44.36	62.87	62.22	61.84	7.25	57.15	55.81	54.10	7.32	61.90	60.80	59.96
	Ours	**94.55**	**58.19**	**57.42**	**56.02**	**93.69**	**63.60**	**63.35**	**63.32**	**57.09**	**57.56**	**56.70**	**55.41**	**58.62**	**62.30**	**61.44**	**60.91**
4	Th	4.90	56.84	56.01	53.86	5.00	62.66	61.94	61.24	2.25	57.02	55.64	53.88	2.30	61.78	60.66	59.75
	VQ	1.91	56.77	55.99	53.94	1.97	62.62	61.91	61.22	1.66	57.03	55.70	53.91	1.55	61.78	60.62	59.73
	[11]	16.19	57.11	56.21	54.20	16.52	62.81	62.14	61.58	4.51	57.15	55.77	54.01	4.52	61.81	60.69	59.77
	Ours	**92.18**	**58.52**	**57.79**	**56.22**	**91.27**	**64.20**	**63.95**	**63.63**	**49.43**	**57.75**	**56.79**	**55.50**	**50.80**	**62.43**	**61.65**	**61.01**

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

		Triplet				Npairs			
	Method	SUF	Pr@1	Pr@4	Pr@16	SUF	Pr@1	Pr@4	Pr@16
k_s	Metric	1.00	10.90	9.39	7.45	1.00	15.73	13.75	11.08
1	LSH	164.25	8.86	7.23	5.04	112.31	11.71	8.98	5.56
	DCH	140.77	9.82	8.43	6.44	220.52	13.87	11.77	8.99
	Th	18.81	10.20	8.58	6.50	1.74	15.06	12.92	9.92
	VQ	146.26	10.37	8.84	6.90	451.42	15.20	13.27	10.96
	[11]	221.49	**11.00**	**9.59**	7.83	478.46	16.95	15.27	13.06
	Ours	**590.41**	10.91	9.58	**7.85**	**952.49**	**17.00**	**15.53**	**13.54**
2	Th	6.33	10.82	9.30	7.32	1.18	15.70	13.69	10.96
	VQ	32.83	10.88	9.33	7.39	116.26	15.62	13.68	11.15
	[11]	60.25	11.10	9.64	7.73	116.61	16.40	14.49	12.00
	Ours	**533.86**	**11.14**	**9.72**	**7.96**	**1174.35**	**17.22**	**15.57**	**13.63**
3	Th	3.64	10.87	9.38	7.42	1.07	15.73	13.74	11.07
	VQ	13.85	10.90	9.38	7.44	55.80	15.74	13.74	11.12
	[11]	27.16	11.20	9.55	7.60	53.98	16.24	14.32	11.73
	Ours	**477.86**	**11.21**	**9.72**	**7.94**	**1297.98**	**17.09**	**15.37**	**13.39**

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 *test*, 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. 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.

	Triplet			Npairs		
	Cifar-100 train	Cifar-100 test	ImageNet val	Cifar-100 train	Cifar-100 test	ImageNet val
LSH	62.94	53.11	37.90	43.80	37.45	36.00
DCH	86.11	68.88	45.55	80.74	65.62	50.01
Th	68.20	54.95	31.62	51.46	44.32	15.20
VQ	76.85	62.68	45.47	80.25	66.69	53.74
[11]	89.11	68.95	48.52	84.90	68.56	55.09
Ours	**89.95**	**69.64**	**61.21**	**86.80**	**71.30**	**65.49**

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

	Method	SUF	Pr@1	Pr@4	Pr@16
k_s	Metric	1.00	21.55	19.11	16.06
1	LSH	33.75	18.49	15.50	11.14
	Th	10.98	20.25	17.22	13.66
	VQ-train	54.30	20.15	18.10	14.85
	VQ-test	57.44	20.59	18.31	15.32
	[11]	56.35	21.35	18.49	15.32
	Ours	**78.23**	**21.46**	**18.88**	**15.67**
2	Th	4.55	21.27	18.86	15.68
	VQ-train	15.29	21.51	19.03	15.88
	VQ-test	16.43	21.58	18.93	15.94
	[11]	15.99	**22.12**	**19.21**	**15.95**
	Ours	**71.14**	**22.12**	18.63	15.34
3	Th	2.79	21.53	19.11	15.99
	VQ-train	7.80	21.56	19.11	16.03
	VQ-test	8.20	21.58	19.09	16.06
	[11]	7.24	**22.18**	**19.40**	**16.10**
	Ours	**84.04**	21.97	18.87	15.56

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_{test}.

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

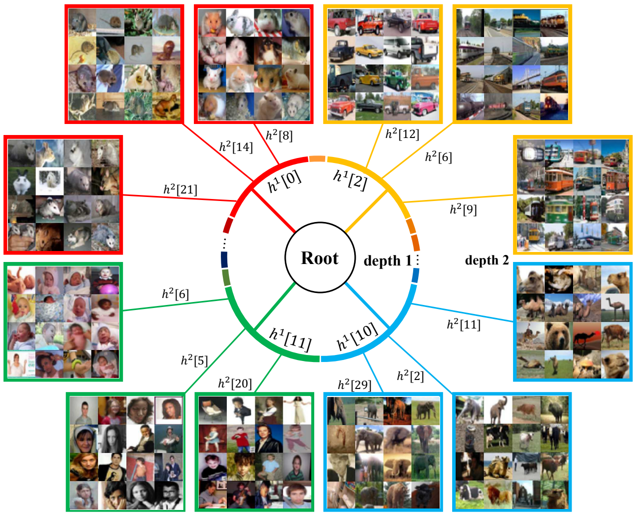

Figure 3: Visualization of the examples mapped by our trained three level hash codes $[\mathbf{h}^{(1)}, \mathbf{h}^{(2)}]$ on Cifar-100. Each parent node (denoted as depth 1) is color coded in red, yellow, blue, and green in *cw* order. Each color coded box (denoted as depth 2) shows examples of the 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 32×32 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 $1298\times$ 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 $C_{\text{train}} \cap C_{\text{test}} = \emptyset$. Section D in the supplementary material shows the class names in C_{train} and C_{test}. We use the images downsampled to 32×32 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

[1] P. Agrawal, R. Girshick, and J. Malik. Analyzing the performance of multilayer neural networks for object recognition. In *ECCV*, 2014. 1, 6

[2] S. Bell and K. Bala. Learning visual similarity for product design with convolutional neural networks. In *SIGGRAPH*, 2015. 1, 2

[3] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 2001. 3

[4] J. Bromley, I. Guyon, Y. Lecun, E. Sackinger, and R. Shah. Signature verification using a "siamese" time delay neural network. In *NIPS*, 1994. 1

[5] M. Bucher, S. Herbin, and F. Jurie. Improving semantic embedding consistency by metric learning for zero-shot classification. In *ECCV*, 2016. 1, 2

[6] Y. Cao, M. Long, B. Liu, and J. Wang. Deep cauchy hashing for hamming space retrieval. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 6

[7] Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen. Deep quantization network for efficient image retrieval. In *AAAI*, 2016. 2

[8] P. Chrabaszcz, I. Loshchilov, and F. Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017. 8

[9] R. Hadsell, S. Chopra, and Y. Lecun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006. 1

[10] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *CVPR*, 2008. 6

[11] Y. Jeong and H. O. Song. Efficient end-to-end learning for quantizable representations. In *ICML*, 2018. 1, 2, 3, 6, 7, 8

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[13] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-100 (canadian institute for advanced research). 2009. 1, 6, 7

[14] Q. Li, Z. Sun, R. He, and T. Tan. Deep supervised discrete hashing. In *NIPS*, 2017. 2

[15] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013. 6

[16] S. Liu and H. Lu. Learning deep representations with diode loss for quantization-based similarity search. In *IJCNN*, 2017. 2

[17] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge university press, 2008. 6

[18] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 1995. 8

[19] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov. Hamming distance metric learning. In *NIPS*, 2012. 2

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015. 1, 6, 7, 8

[21] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015. 1, 3, 5, 6

[22] O. Sener, H. O. Song, A. Saxena, and S. Savarese. Learning transferrable representations for unsupervised domain adaptation. In *NIPS*, 2016. 2

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

[24] H. O. Song, S. Jegelka, V. Rathod, and K. Murphy. Deep metric learning via facility location. In *CVPR*, 2017. 1

[25] H. O. Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *CVPR*, 2016. 1

[26] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014. 1

[27] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen. A survey on learning to hash. *arXiv preprint arXiv:1606.00185*, 2016. 1, 2, 6

[28] X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015. 2

[29] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, 2014. 2

[30] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015. 6

[31] Y. Yuan, K. Yang, and C. Zhang. Hard-aware deeply cascaded embedding. In *ICCV*, 2017. 1, 2

[32] A. Zhai, D. Kislyuk, Y. Jing, M. Feng, E. Tzeng, J. Donahue, Y. L. Du, and T. Darrell. Visual discovery at pinterest. In *Proceedings of the 26th International Conference on World Wide Web Companion*, 2017. 1, 6

[33] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, 2015. 2