SHOE: Sibling Hashing with Output Embeddings

Sravanthi Bondugula, Varun Manjunatha, Larry S. Davis, David Doermann
sravb@cs.umd.edu, varunm@cs.umd.edu, lsd@umiacs.umd.edu, doermann@umd.edu
University of Maryland, College Park, MD, 20742, USA

ABSTRACT
We present a supervised binary encoding scheme for image retrieval that learns projections by taking into account similarity between classes obtained from output embeddings. Our motivation is that binary hash codes learned in this way improve the visual quality of retrieval results by ranking related (or “sibling”) class images before unrelated class images. We employ a sequential greedy optimization that learns relationship aware projections by minimizing the difference between inner products of binary codes and output embedding vectors. We develop a joint optimization framework to learn projections which improve the accuracy of supervised hashing over the current state of the art with respect to standard and sibling evaluation metrics. We further obtain discriminative features learned from correlations of kernelized input CNN features and output embeddings, which significantly boosts performance. Experiments are performed on three datasets: CUB-2011, SUN-Attribute and ImageNet ILSVRC 2010, where we show significant improvement in sibling performance metrics over state-of-the-art supervised hashing techniques, while maintaining performance with respect to standard metrics.

Categories and Subject Descriptors
H.3 Information Storage and Retrieval: Content Analysis and Indexing, Information Search and Retrieval

General Terms
Hashing, Retrieval, Sibling classes

Keywords
Supervised Hashing, Sibling Hashing, Output Embeddings

1. INTRODUCTION
Supervised Image Hashing [1, 8, 7] is a method which maps images belonging to the same class to similar binary codes. Recent approaches include Supervised Hashing with Kernels [8], which construct binary projections sequentially by greedily minimizing the difference between Hamming distances and similarity of pairs of data points; and FastHash [7] which employs Boosted Decision Trees as hash functions and utilizes a GraphCut based method for binary code inference. In this work, we develop a new approach to Supervised Hashing, which we motivate with the example shown in Figure 1. Consider an image retrieval problem that involves a database of animals and a query image of a leopard. Now consider the following three scenarios:

1. If the retrieval algorithm returns images of leopards, we can deem the result to be absolutely satisfactory.
2. If the retrieval algorithm returns images of dolphins, whales or sharks, we consider the results to be absolutely unsatisfactory because not only are dolphins not leopards, they do not look anything at all like leopards.
3. If the retrieval algorithm returns the image of a jaguar or a tiger, we would be reasonably satisfied with the results. Although a jaguar is not the same as a leopard, it is semantically similar to one.

In this example, leopards, dolphins, whales, sharks, jaguars and tigers all belong to different categories. However, some of these categories are more closely related to each other than to other categories. Animals which fall under the “big cat” (Panthera) genus are related to each other, as are the large aquatic vertebrates like dolphins, sharks and whales. We designate the related categories as “siblings”. The question now becomes: how can we construct hash functions...
that rank sibling class images ahead of unrelated class images? Traditional supervised hashing algorithms like [8, 7] use discrete binary labels to compute similarity between classes. We, on the other hand, consider information outside the model's parameters and use continuous similarity scores.

To study the relationships between categories, Weinberger et al. [11] suggested the concept of “output embeddings” - vector representations of category information in Euclidean space. There has been extensive work on “output embeddings”, which are vector-space representations of images [5], but less work has been done on output embeddings, which map similar category labels to similar vectors in Euclidean space. For example, in an output embedding space of animals, we would expect to have embeddings for labels so that chimpanzees, orangutans and gorillas are near each other as are leopards, cheetahs, tigers and jaguars. Output embeddings can be data-dependent, such as [6, 10] which uses attributes or Linnean taxonomies to obtain vector representations for classes, or data-independent, like [4] where the embeddings for each class are random vectors containing ±1.

Our method, which uses output embeddings to construct hash codes in a supervised framework is called SHOE: Sibling Hashing with Output Embeddings. Our motivation for doing this is the following: it is our belief that in the case of Supervised Hashing, a more desirable algorithm will retrieve images of sibling classes ahead of images of unrelated classes while maintaining the retrieval performance for images of the same class. This improves the overall visual quality of retrieved results.

The contributions of our paper are as follows: 1) To the best of our knowledge, our approach is the first to introduce the problem of learning supervised hash functions using the modality of output embeddings. 2) We propose a joint learning method to solve the above problem, and experimentally validate our method. 3) We propose two new evaluation criteria - “sibling metrics” and “weighted sibling metrics”, for gauging the efficacy of our method. 4) We significantly boost retrieval performance by applying Canonical Correlation Analysis on input features, and learn hash functions using output embeddings on these features.

The remainder of this paper is arranged as follows. In Section 2 we describe our hashing framework, and carry out experiments in Section 3 and conclude in Section 4.

2. METHOD

2.1 SHOE

Given a training set $M = \{ (x_1, y_1), \ldots, (x_N, y_N) \}$ of $N$ (image, label) pairs with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, let $\phi : \mathcal{X} \rightarrow \mathcal{X} \subset \mathbb{R}^d$ be the input embedding function and $\psi : \mathcal{Y} \rightarrow \bar{\mathcal{Y}} \subset \mathbb{R}^d$ be the output embedding function. We wish to learn binary codes $b_i, b_j$ of length $c$ (i.e., $b_i \in \{-1,1\}^c$) such that for pairs of training images, the Hamming distance ($d_H(b_i, b_j)$) between the codes preserves the distance between their class labels (given by their corresponding output embedding vectors). In other words, for a given query image, retrieved results of sibling(unrelated) classes ought to be ranked higher(lower). Utilizing the equivalence between inner product of binary codes and Hamming distances [8], $2d_H(b_i, b_j) = c - b_i^T b_j$, where $b_i^T b_j$ is the inner product of the binary codes $b_i$ and $b_j$ and the traditional hyperplane hash function definition $h(x) = \textrm{sgn}(w^T x)$, $w$ is the projection vector), we obtain the following objective function:

$$
\min_{\theta} \frac{N}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \left( \frac{1}{c} \sum_{l=1}^{c} h_l(\phi(x_i)) h_l(\phi(x_j)) - \bar{\psi}(y_i)^T \bar{\psi}(y_j) \right)^2
$$

where $H = \{ h_l | l = 1, \ldots, c \}$ denote the $c$ hash functions to be learned. Equivalently, we want to learn $c$ projection vectors ($W = [w_1, \ldots, w_c] \in \mathbb{R}^{c \times d}$). Note that the normalized inner product of binary codes lies between $-1$ and $1$ and the dot product of the unit normalized output embedding vectors (denoted by $o_{ij} = \bar{\psi}(y_i)^T \bar{\psi}(y_j)$) also lies between $-1$ and $1$ and therefore, the equivalence. The objective ensures that the learned binary codes preserve the similarity between output embeddings, which is required for our goal of ranking related neighbors before farthest neighbors. This is similar to the KSH [8] objective function, except that KSH assumes that $o_{ij}$ takes only values $1(-1)$ for similar(dissimilar) pairs defined with semantic information. Their work also accommodates the definition of the $o_{ij}$ for related pairs but only for metric neighbors. Our work is different from theirs, as we emphasize the learning of binary codes that preserve the similarity between the classes, given by $o_{ij}$. Regardless of the definition of $o_{ij}$, our optimization is similar, and so we employ a similar sequential greedy optimization. We refer the reader to [8] for further details.

2.2 SHOE Revisited

We observe that using similarity directly from output embeddings reduces the performance of retrieving same class images, while improving the retrieval of sibling class images. Let $\theta$ measure the similarity between a class and a related class and we wish to learn such a similarity value that maximizes both the performance of same class and related class pairs. To do this, we split the objective function of Equation (1) into three parts - for identical classes, sibling classes and unrelated classes, respectively. We also empirically observe that precision and recall metrics improve for negative values of $\theta$. Therefore, we add regularizer term $\lambda ||\theta + 1||^2$ to the objective function, which becomes small when $\theta$ lies close to -1. For easier notation, we denote $h_l = h_l(\phi(x_i))$. Our modified objective function now becomes:

$$
\min_{W, \theta} \sum_{i,j}^{N} \sum_{c \in \text{same class}} \left( \frac{1}{c} \sum_{l=1}^{c} h_l h_{ij} - 1 \right)^2 + \sum_{i,j}^{N} \sum_{c \in \text{sibling class}} \left( \frac{1}{c} \sum_{l=1}^{c} h_l h_{ij} - \theta \right)^2 + \sum_{i,j}^{N} \sum_{c \in \text{related class}} \left( \frac{1}{c} \sum_{l=1}^{c} h_l h_{ij} + 1 \right)^2 + \lambda \parallel \theta + 1 \parallel^2
$$

We solve the above objective function in a two-step process, where we first learn the hash functions and then the similarity value $\theta$. Let $H_{ij} = \frac{1}{c} \sum_{l=1}^{c} h_l h_{ij}$ and given $H_{ij}$, we compute the derivative of Equation (2) w.r.t $\theta$, set it to 0 and solve for each $\theta$. We obtain:

$$
\theta = \frac{\sum_{i,j}^{N} \sum_{c \in \text{sibling class}} (H_{ij} - \lambda)}{n_{\text{pair}} + \lambda}
$$

where $n_{\text{pair}}$ is the number of sibling pairs in the training data. We have thus obtained a closed form solution for the
optimal $\theta$. However, we cannot calculate $\theta$ directly as we do not learn all the bits at once. Therefore, we employ a two step alternate optimization procedure that first learns the bits and then an approximate $\theta_1$ value calculated from the previously learned bits. For the first iteration, we use an initial $\theta_0$ value, computed from the similarity of the output embeddings. The two step optimization procedure for learning the $t^{th}$ hash function is:

1. Step 1: We optimize for Equation (1), keeping $\theta_{t-1}$ constant and updating the projection vector $W$, thus learning hash-code bits $h_t(\phi(x_i))$.

2. Step 2: We keep the hash-code bits $h_t(\phi(x_i))$ constant and learn $\theta_t$ for each image pair using Equation (3).

2.3 Evaluation Criteria

Standard metrics like precision, recall and mAP defined for semantic neighbors are not sufficient to evaluate the retrieval of the sibling class images. To measure this, we define sibling metrics. Let $R_y : (y_i, y_j) \rightarrow \text{rank}$ return the rank of class $y_j$ for a query class $y_i$, $0 \leq \text{rank} \leq L$, where $L$ is the number of classes and $\text{Sib}_m(y_i, y_j, R_y) \rightarrow \frac{m - R_y(y_i, y_j)}{m}$ gives the sibling weight between the classes, when only $m$ sibling classes are considered. We use $m = 5$. The ranking $R_y$ is computed by sorting the distance between the output embedding vectors $\psi(y_i)$ and $\psi(y_j)$, $y_j \in Y \setminus y_i$. The sibling metrics for a query $q$ are defined as ($y_q$ refers to the class of the $l^{th}$ retrieved image):

$$w_s^{\text{precision}}@k = \frac{\sum_{i=1}^{k} \text{Sib}_m(y_q, y_q, R_y)}{k}$$

(4)

$$w_s^{\text{recall}}@k = \frac{\sum_{i=1}^{k} \text{Sib}_m(y_q, y_q, R_y)}{\sum_{p=1}^{L} \sum_{i=1}^{m} \|\text{Sib}_m(y_q, y_q, R_y)\|}$$

(5)

$$s_{w_{\text{AP}}} = \sum_{k=1}^{N} s_{w_s^{\text{precision}}@k} \times \Delta s_{w_{\text{recall}}@k}$$

(6)

$$s_{w_{\text{mAP}}} = \frac{\sum_{k=1}^{Q} s_{w_{\text{AP}}}@k}{Q}$$

(7)

3. EXPERIMENTS

We evaluate our method on Caltech-UCSD-2011 Birds (CUB) [12], the SUN Attribute [9] and ILSVRC 2010 [2] datasets. We extract CNN features of the fc7 layer[5] and further kernelize them to take the form $\langle \kappa(x, x_1), \ldots, \kappa(x, x_p) \rangle$ where $\kappa$ is a radial basis kernel, and $p$ is the cardinality of a subset of training sample, designated as “anchor points”.

Output Embeddings/Train-Test Partitions: For CUB dataset, we use the 312 mean-centered real-valued attribute vectors as output embeddings and only select 2000 samples for training and $p = 300$ anchor points. We use the whole train set for retrieval and all the test images as queries. For ImageNet experiments, we obtain output embeddings using the method of Tsotschantaridis[10]. We uniformly select 2 images per class as a test set and use the rest of the 1.2 million images as retrieval set. We select 5000 images and
3000 anchor points for training. For SUN dataset, we use the 102 real valued attributes as output embeddings. We partition the dataset into equal retrieval and test sets and only use a subset of 3555 and 1434 images as training and anchor point images respectively. For all the datasets, we define the ground truth using class labels.

**Results:** Two variants of SHOE are used - SHOE(E), which uses raw output embeddings as in Section 2.1, and SHOE(L), which is learned using the method in Section 2.2. We compare with the state-of-the-art methods\(^1\) : KSH[8], FastHash[7], ITQ[3] and LSH[1] in Supervised Hashing literature and present the results in Figure 2.

In Figure 2, the rows represent weighted sibling, sibling and mAP metrics. The first two columns represent experiments without and with CCA projections on the CUB dataset, while the latter two columns represent the same on the SUN Attribute dataset. We notice that without CCA projections, we outperform the baselines on standard metrics, and comfortably outperform on sibling metrics. With output embedded CCA projections, our method performs as well as the best baseline on standard metrics, and comfortably outperforms on sibling metrics. CCA projections significantly improve the performance for SHOE, KSH and ITQ, while it lowers the performance for FastHash and LSH. Results on the ILSVRC2010 dataset in Figure 3 show that with CCA features, SHOE surpasses KSH on recall@10K and sibling precision metrics, while performing as well as KSH on precision@50 and mAP. Qualitative results which compare SHOE and KSH can be found in Figure 4.

**4. CONCLUSION**

The key idea in our paper is to learn binary codes that respect the relationship between classes. We utilize output embeddings to define the similarity between classes and obtain binary codes that preserve class order. To the best of our knowledge, ours is the first work to do so. We devised a method to learn class similarity jointly with the hash function, along with new metrics for their evaluation. It is our belief that this scheme improves overall visual quality of the retrieval system, and we have validated this experimentally with sibling metrics. Our method, called SHOE, achieves state-of-the-art image retrieval results with and without CCA projections over multiple datasets.

**5. REFERENCES**


\(^{1}\)We use their publicly available implementations

---

**Figure 3:** Retrieval on ILSVRC2010 dataset comparing SHOE with the state-of-the-art hashing techniques. Table reports the performance for 256 bits. We use Kernelized CNN features with CCA.

**Figure 4:** The first query is of an ovenbird. SHOE retrieves more ovenbirds than KSH. The second query is of a Brewer black-bird. Neither SHOE nor KSH retrieve Brewer black-birds. However, SHOE returns ravens(sibling classes of Brewer black-birds), whereas KSH retrieves piliated woodpeckers(unrelated to black-birds). Here, blue borders represent sibling classes.